



DeckBuild

User's Manual

SILVACO, Inc.

4701 Patrick Henry Drive, Bldg. 2
Santa Clara, CA 95054

Phone (408) 567-1000

Web: www.silvaco.com

April 22, 2010

DeckBuild
User's Manual
Copyright 2010

SILVACO, Inc.
4701 Patrick Henry Drive, Bldg. #2
Santa Clara, CA 95054

Phone: (408) 567-1000
Web: www.silvaco.com

The information contained in this document is subject to change without notice.

SILVACO, Inc. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE.

SILVACO, Inc. shall not be held liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

This document contains proprietary information, which is protected by copyright laws of the United States. All rights are reserved. No part of this document may be photocopied, reproduced, or translated into another language without the prior written consent of SILVACO, Inc.

ACCUCELL, ACCUCORE, ACCUMODEL, ACCUTEST, ATHENA, ATHENA 1D, ATLAS, BLAZE, C-INTERPRETER, CATALYSTDA, CATALYSTAD, CELEBRITY, CELEBRITY C++, CIRCUIT OPTIMIZER, CLARITYRLC, CLEVER, DECKBUILD, DEVEDIT, DEVEDIT3D, DEVICE3D, DISCOVERY, EDA OMNI, EDIF WRITER, ELITE, EXACT, EXPERT, EXPERTVIEWS, FERRO, GATEWAY, GIGA, GIGA3D, GUARDIAN, GUARDIAN DRC, GUARDIAN LVS, GUARDIAN NET, HARMONY, HIPEX, HIPEX-C, HIPEX-NET, HIPEX-RC, HYPERFAULT, LASER, LED, LISA, LUMINOUS, LUMINOUS3D, MAGNETIC, MAGNETIC3D, MASKVIEWS, MC DEPO/ETCH, MC DEVICE, MC IMPLANT, MERCURY, MIXEDMODE, MIXEDMODE3D, MOCASIM, MODELLIB, NOISE, NOMAD, OLED, OPTOLITH, ORGANIC DISPLAY, ORGANIC SOLAR, OTFT, PROMOST, QUANTUM, QUANTUM3D, QUEST, REALTIMEDRC, RESILIENCE, S-PISCES, S-SUPREM3, S-SUPREM4, SCOUT, SDDL, SFLM, SiC, SILOS, SMARTLIB, SMARTSPICE, SMARTSPICE SEE, SMARTSPICERF, SMARTVIEW, SIMULATION STANDARD, SOLVERLIB, SPAYN, SPDB, SPIDER, SPRINT, STELLAR, TCAD DRIVEN CAD, TCAD OMNI, TFT, TFT3D, THERMAL3D, TONYPLOT, TONYPLOT3D, TWISTER, TWISTERFP, VCSEL, UTMOST, UTMOST III, UTMOST IV, UTMOST IV- FIT, UTMOST IV-MEASURE, UTMOST IV- OPTIMIZATION, VICTORY, VICTORYCELL, VICTORYDEVICE, VICTORYPROCESS, VICTORYSTRESS, VIRTUAL WAFER FAB, VWF, VWF AUTOMATION TOOLS, VWF INTERACTIVE TOOLS, VWF MANUFACTURING TOOLS, and VYPER are trademarks of SILVACO, Inc.

All other trademarks mentioned in this manual are the property of their respective owners.

Copyright © 1984 - 2010, SILVACO, Inc.

How to Read this Manual

Style Conventions		
Font Style/Convention	Description	Example
•	This represents a list of items or terms.	<ul style="list-style-type: none"> Bullet A Bullet B Bullet C
1. 2. 3.	This represents a set of directions to perform an action.	To open a door: <ol style="list-style-type: none"> Unlock the door by inserting the key into keyhole. Turn key counter-clockwise. Pull out the key from the keyhole. Grab the doorknob and turn clockwise and pull.
→	This represents a sequence of menu options and GUI buttons to perform an action.	File→Open
Courier	This represents the commands, parameters, and variables syntax.	HAPPY BIRTHDAY
New Century Schoolbook Bold	This represents the menu options and buttons in the GUI.	File
<i>New Century Schoolbook Italics</i>	This represents the equations.	$abc=xyz$
Note:	This represents the additional important information.	Note: Make sure you save often while running an experiment.
NEW CENTURY SCHOOLBOOK IN SMALL CAPS	This represents the names of the SILVACO products.	ATHENA and ATLAS..

Table of Contents

Chapter 1

Introduction 1-1

1.1: What is DeckBuild 1-1

1.1.1: Purpose 1-1

1.1.2: Features 1-1

Chapter 2

Tutorial 2-1

2.1: QuickStart 2-1

2.1.1: Starting DeckBuild 2-1

2.1.2: Writing a SSUPREM3 Input Deck 2-1

2.1.3: Running A Deck 2-4

2.1.4: Quitting DeckBuild 2-6

Chapter 3

Functions 3-1

3.1: Starting DeckBuild 3-1

3.1.1: Syntax 3-1

3.1.2: Description 3-1

3.1.3: Options 3-1

3.2: DeckBuild Controls 3-4

3.2.1: Main Window Layout 3-4

3.2.2: Text & TTY Subwindows 3-5

3.2.3: Using the Text Subwindow 3-5

3.2.4: Using the TTY Subwindow 3-8

3.3: Main Control 3-10

3.3.1: Control Pad 3-10

3.3.2: Choosing a Simulator 3-10

3.3.3: Simulator Properties 3-11

3.3.4: Start Simulator 3-11

3.3.5: Simulator Controls 3-11

3.3.6: Options Category 3-13

3.3.7: Messages Category 3-15

3.3.8: Formatting Category 3-16

3.3.9: Arguments Category 3-16

3.4: Execution Control 3-17

3.4.1: Execution Concepts 3-17

3.4.2: Execution Control Buttons 3-18

3.4.3: Stepping Through and Running the Deck 3-18

3.4.4: Setting and Clearing Breakpoints 3-18

3.4.5: Setting the Current Line 3-18

3.4.6: Pausing, Stopping, and Restarting the Simulator 3-19

3.4.7: Initializing the Simulator 3-19

3.5: Commands 3-20

3.5.1: Deck Writing Paradigm 3-20

3.5.2: Parsing the Deck 3-20

3.5.3: Process Simulators 3-21

3.5.4: Clever 3-22

3.6: Tools	3-23
3.6.1: Starting TonyPlot	3-23
3.6.2: Starting Maskviews	3-24
3.6.3: Starting Text Editor	3-27
3.6.4: Starting Manager	3-27
3.7: History	3-28
3.7.1: Overview	3-28
3.7.2: History Control	3-28
3.8: Auto Interfacing	3-30
3.8.1: Scenario	3-30
3.9: IC Layout Interface	3-33
3.9.1: Creating a Generic Deck	3-33
3.9.2: Regions	3-34
3.9.3: Rules of Thumb	3-36
3.9.4: Mask Bias, Misalignment, and Delta CD	3-36
3.9.5: Using DevEdit with IC Layout	3-37
3.10: UTMOST Interface	3-38
3.10.1: Setting Up An UTMOST Input Deck	3-38
3.11: SmartSpice Interface	3-46
3.12: Internal Interface	3-47
3.13: Remote Simulation	3-48
3.13.1: Remote Options	3-48
3.13.2: Troubleshooting	3-48

Chapter 4

Statements	4-1
4.1: Overview	4-1
4.1.1: DeckBuild Commands	4-1
4.2: ASSIGN	4-2
4.3: AUTOELECTRODE	4-5
4.4: DEFINE and UNDEFINE	4-6
4.5: EXTRACT	4-8
4.6: GO	4-9
4.7: IF, ELSE and IF.END	4-11
4.8: LOOP, L.END and L.MODIFY	4-12
4.9: MASK	4-14
4.10: MASKVIEWS	4-16
4.11: SET	4-17
4.12: SOURCE	4-19
4.13: STMT	4-20
4.14: SYSTEM	4-23
4.15: TONYPLOT	4-24

Chapter 5

Extract	5-1
5.1: Overview	5-1
5.2: Process Extraction	5-2
5.2.1: Entering a Process Extraction Statement	5-4
5.2.2: Extracting a Curve	5-5
5.3: Customized Extract Statements	5-7
5.3.1: Extract Syntax	5-7
5.3.2: DEFAULTS	5-20
5.3.3: Examples of Process Extraction	5-21
5.4: Device Extraction	5-29
5.4.1: The Curve	5-29
5.4.2: Curve Manipulation	5-31
5.4.3: BJT Example	5-32
5.5: General Curve Examples	5-33
5.5.1: Curve Creation	5-33
5.5.2: Min Operator with Curves	5-33
5.5.3: Max Operator with Curves	5-33
5.5.4: Ave Operator with Curves	5-33
5.5.5: X Value Intercept for Specified Y	5-33
5.5.6: Y Value Intercept for Specified X	5-33
5.5.7: Abs Operator with Axis	5-34
5.5.8: Min Operator with Axis Intercept	5-34
5.5.9: Max Operator with Axis Intercept	5-34
5.5.10: Second Intercept Occurrence	5-34
5.5.11: Gradient at Axis Intercept	5-34
5.5.12: Axis Manipulation with Constants	5-34
5.5.13: X Axis Interception of Line Created by Maxslope Operator	5-34
5.5.14: Y Axis Interception of Line Created by Minslope Operator	5-35
5.5.15: Axis Manipulation Combined with Max and Abs Operators	5-35
5.5.16: Axis Manipulation Combined with Y Value Intercept	5-35
5.5.17: Derivative	5-35
5.5.18: Data Format File Extract with X Limits	5-35
5.5.19: Impurity Transform against Depth	5-35
5.6: MOS Device Tests	5-36
5.7: Extracted Results	5-37
5.7.1: Units	5-37
5.8: Extract Features	5-38
5.8.1: Extract Name	5-38
5.8.2: Variable Substitution	5-38
5.8.3: Min and Max Cutoff Values	5-38
5.8.4: Multi-Line Extract Statements	5-39
5.8.5: Extraction and the Database (VWF)	5-39
5.9: QUICKBIP Bipolar Extract	5-40
5.10: Using Extract with ATLAS	5-43

Chapter 6

Optimizer	6-1
6.1: Overview	6-1
6.1.1: Features	6-1
6.1.2: Terminology	6-1
6.2: Using The Optimizer	6-2
6.2.1: Overview	6-2
6.2.2: The Optimizer Window	6-2
6.2.3: Optimization Modes	6-3
6.2.4: Optimizing	6-3
6.3: Parameters	6-4
6.3.1: Adding Parameters	6-4
6.3.2: Deleting Parameters	6-6
6.3.3: Editing A Parameter	6-7
6.3.4: Linked Parameters	6-9
6.3.5: Parameter Defaults	6-10
6.3.6: Copying Parameters To The Deck	6-10
6.3.7: Enabling/Disabling Parameters	6-10
6.3.8: Folding Columns	6-11
6.4: Targets	6-12
6.4.1: Adding A Target	6-12
6.4.2: Deleting A Target	6-16
6.4.3: Editing A Target	6-17
6.4.4: Enabling/Disabling Target	6-18
6.4.5: Folding Columns	6-19
6.5: Setup	6-20
6.5.1: Overview	6-20
6.5.2: Editing Setup Values	6-20
6.5.3: Saving The Setup	6-20
6.6: Graphics	6-21
6.6.1: Overview	6-21
6.7: Results	6-23
6.7.1: Overview	6-23
6.7.2: Sensitivity	6-23
6.8: Worksheet Editing	6-24
6.8.1: Overview	6-24
6.8.2: Numeric Values	6-24
6.8.3: Selecting Rows	6-24
6.8.4: Mouseless Operation	6-25
6.9: File I/O	6-26
6.9.1: Overview	6-26
6.9.2: Creating A New File	6-26
6.9.3: Working With An Existing File	6-27
6.9.4: Saving The Existing File	6-27
6.10: Printing	6-28
6.10.1: Overview	6-28
6.11: Optimization Tuning	6-30

Appendix A

Models and Algorithms A-1

A.1: Introduction	A-1
A.1.1: Physical Models	A-1
A.2: Concentration Dependent Mobility	A-2
A.3: Field Dependent Mobility Model	A-3
A.4: Sheet Resistance Calculation	A-4
A.5: Threshold Voltage Calculation	A-5
A.5.1: Breakdown Voltage Calculation	A-6

Appendix B

DBInternal B-1

B.1: DBInternal	B-1
B.1.1: Example	B-1
B.2: The Template File	B-3
B.3: The Experiment File	B-4
B.3.1: Load command	B-4
B.3.2: Experiment command	B-4
B.3.3: Save Command	B-4
B.4: Technical Details	B-5
B.5: DBInternal Commands	B-6
B.5.1: doe	B-6
B.5.2: endsave	B-8
B.5.3: log	B-8
B.5.4: monte_carlo	B-8
B.5.5: no_exec	B-10
B.5.6: option	B-11
B.5.7: save	B-11
B.5.8: sweep	B-12
B.6: DBIT	B-14
B.6.1: The General Tab	B-15
B.6.2: The Matrix Tab	B-17
B.6.3: The Command Menu	B-18

Appendix C

Environment Variables C-1

C.1: List of Variables	C-1
------------------------------	-----

Appendix D

Error Messages D-1

D.1: Text Subwindow Error Messages	D-1
D.2: TTY Subwindow Error Messages	D-2

This page is intentionally left blank.

1.1: What is DeckBuild

DECKBUILD is an interactive, graphic runtime environment for developing process and device simulation input decks. It consists of a window for input deck creation and editing, a window for simulator output and control, and a set of popups for each simulator that provide full language and run-time support.

1.1.1: Purpose

DECKBUILD is an extremely powerful and flexible tool that is easy to use and provides many automated features that previously required user operation. Among these features is DECKBUILD's ability to generate error-free simulator syntax driven from user-friendly popup windows. This feature allows for transparent transition from one simulator to another, automatic definition of mesh and mask information, and application of built-in measurement (extraction) facilities. Before DECKBUILD, these tasks often required user intervention and were extremely time consuming. By automating these tasks, DECKBUILD allows you to concentrate on the real work at hand: accurate simulation.

1.1.2: Features

DECKBUILD also offers several powerful facilities never before available. One of these facilities, the global optimizer, allows optimization across an entire input deck even between different simulators. For example, varying an implant dose in SSUPREM3 and a diffusion time in ATHENA permits optimizing against a Vt curve simulated with ATLAS. DECKBUILD also provides a seamless integration with DEVEDIT and its adaptive meshing capabilities. In addition, the UTMOST interface allows SILVACO's parameter extraction package UTMOST III to load data from one of more device simulation runs and perform SPICE model parameter extraction. DECKBUILD offers real flexibility with the ability to use UNIX system commands within simulation decks and the added feature of executing simulations on remote hosts while DECKBUILD is running locally.

One new feature is the communication interfaces to EXACT. EXACT uses the new input/output pipe command-line options to send a simulation deck for execution. EXACT then receive extracted results from DECKBUILD.

Note: Pipe in this context means a conduit that transfers data between two program.

DECKBUILD also contains many other convenience features:

- A built-in tool palette allows interactive plotting of the current structure.
- Instant substitution of a different cut-line set from the layout editor.
- Full interactive control of the simulator, including a history function that allows you to back up in the deck and try again.
- Interactive cut-and-paste to either the simulator or the text editor.
- A display in the input deck of the currently executing line and other features.

Simulators

Many simulators are available in the DECKBUILD Library and most are supported by a complete set of interactive popup windows. By selecting or moving various items on each popup, you can easily generate correct syntax. On-line context-sensitive help is also available. A deck is built by going through each desired popup and clicking on a **WRITE** button. This causes syntax to appear in the text editor. The deck can be saved and retrieved for later use. The popups have the additional feature of input-deck parsing. To do this, highlight a section of the input deck and choosing **Parse Deck**. All appropriate popups will then re-configure themselves to reflect the syntax. For example, if you highlight an ATHENA IMPLANT statement and press the **Parse Deck**, the ATHENA Implant popup will appear. This popup would reflect the values in the highlighted syntax.

For manual deck editing, DECKBUILD has a built-in text editor. The text editor allows easy point-and-click editing, cut and paste to and from any other window, find/replace, multiple scroll views, and other features.

Autointerface

DECKBUILD allows and encourages concatenating of decks from different simulators. For example, a simulation can start with SSUPREM3 for fast 1-D process simulation, move into ATHENA for 2-D process simulation, and be followed by any number of separate ATLAS device tests. Figure 1-1 shows a schematic of this flow. The entire result is saved as a single input deck.

Notice how process simulation is treated as a serial flow of events, while device simulations are treated as parallel. This is because of the way in which auto interfacing works. At the conclusion of each process run, the simulation results are saved and are used by the next process simulator; several process decks form a serially-linked chain. Device tests always use the last available process result. Auto interfacing is one of the most powerful features in DECKBUILD.

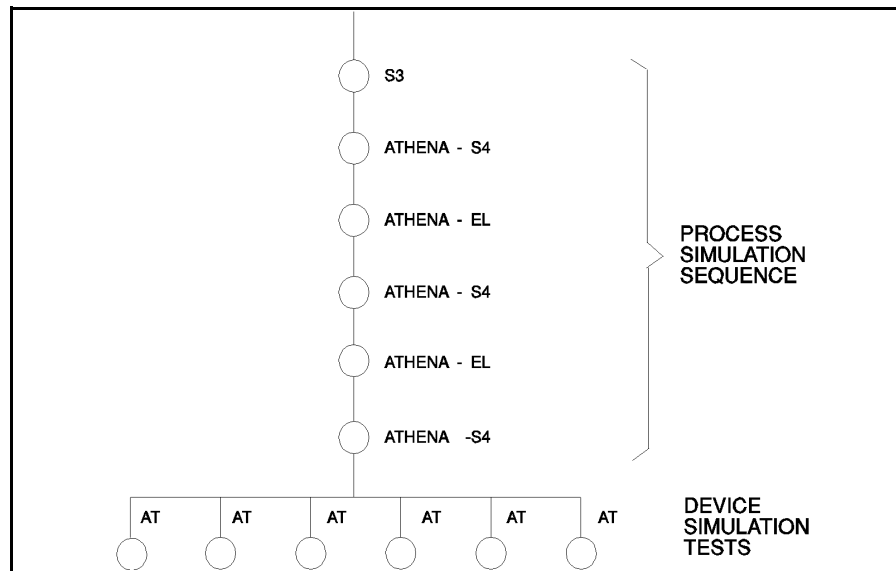


Figure 1-1: DeckBuild Schematic

Execution Control

DECKBUILD provides a diverse set of controls over the running simulation. You can run the entire deck. You can run it one line at a time. You can run the deck until a predefined line is reached or halted immediately after the current command. You can even set multiple break points in the deck by using the **String Monitor** option and by adding a specific comment to search for at the required locations. While the simulation is running, DECKBUILD also highlights the currently executing line in the input deck. The simulator itself can be stopped, started, quit, killed, paused, and unpaused.

One of DECKBUILD's unique features is the **History** function (see Section 3.7: "History"). DECKBUILD remembers each line of the deck as it is executed and saves a structure file after each one. As a result, if a problem is discovered it is unnecessary to re-do the entire deck from the start. For example, if after running part way down an input deck and you discover a missing statement or an erroneous value, you only need to point and click on the line from which to start and click on **Init from History**. DECKBUILD automatically re-loads the saved history file and allows the simulation to continue from that point.

DECKBUILD also allows plotting the current structure. At any point in the deck, click a button and DECKBUILD automatically causes the simulator to save a structure file, then start up SILVACO's post-processing tool using the saved structure as input. This is often useful in conjunction with **History** to aid in fast tuning of a section of input deck. A statement can also be changed then re-executed, and the change is immediately visualized.

Examples and Tutorial

DECKBUILD provides full on-line examples that can be loaded up at the press of a button. These examples provide input decks for actual devices and help when learning about DECKBUILD. Chapter 2: "Tutorial" is a tutorial that explains how to use DECKBUILD to perform a simple simulation.

Advanced Topics

Generic Decks

Most decks have built-in geometric constants that reproduce a single, unchangeable cross-section of a wafer. DECKBUILD's IC layout interface (MASKVIEWS) makes it possible to write a single deck that can be used at any location on a wafer without using hard-coded geometry information. You can create (or read from GDSII or CIF format) device layout and mask layers using MASKVIEWS. Then, create or modify a deck using DECKBUILD to use mask names rather than deposit and etch statements with hard-coded geometry values. Finally after making a cutline using MASKVIEWS, DECKBUILD can simulate that cross-section. You can simulate any cross-section of the device in this manner.

Extraction

DECKBUILD contains built-in extract routines for both process and device parameter extraction. EXTRACT forms a "function calculator" that allows you to combine and manipulate values or entire curves quickly and easily. You can take one of the standard expressions and modify it as appropriate to suit your needs or use the custom extract language to create unique extraction statements specific to the current simulation.

EXTRACT also includes features, such as variable substitution and internal 1D device simulators, QUICKMOS and QUICKBIP for specialized cases of MOS and bipolar electrical measurement. All extracted results are displayed in the DECKBUILD TTY subwindow and stored in a datafile for easy comparisons of different simulations (See Chapter 5: "Extract").

Optimization

A powerful OPTIMIZER is available within DECKBUILD that allows quick and accurate tuning of simulation parameters. Specify any number of input parameters to vary and any number of targets to attain. For example, it is possible to find a target threshold voltage of 0.75 volts by varying gate oxidation time and V_t adjust implant dose.

Optimization parameters may come from any simulator in the simulator library, and targets from any extracted parameter. For example, it is easy to set up a deck that auto interfaces from SSUPREM3 to ATHENA to ATLAS. Then, extracted values can be optimized from I-V curves while using SSUPREM3 or ATHENA diffusion coefficients or both as input parameters. Simple graphical worksheet control with interactive runtime display of the optimization in progress makes the optimizer easy to use. In addition, the OPTIMIZER requires no modification of any kind to the input deck.

2.1: QuickStart

Although DECKBUILD may seem complicated at first, a deck can be built and run using only a few basic controls. If new to DECKBUILD, please follow this tutorial guide. The remainder of the manual can be read for more details on using DECKBUILD after getting familiar with its operation.

2.1.1: Starting DeckBuild

This section explains how to start DECKBUILD, create an input deck, run the deck, do an interactive plot of the results, and save the input deck file to disk. In the following startup instructions, if not running C-Shell, do not use the ampersand (&) at the end of the entered command line. The & tells csh to run DECKBUILD in the background.

Since a SSUPREM3 input deck is built in this tutorial, start DECKBUILD with the default simulator set to SSUPREM3 by entering the command:

```
deckbuild -s3 &
```

The simulator can also be selected or changed after DECKBUILD has been started from the Main Control popup.

In a few moments, DECKBUILD will appear on the screen. If it doesn't or if any error messages appear, refer to the SILVACO INSTALLATION GUIDE to verify that DECKBUILD was installed correctly.

2.1.2: Writing a SSUPREM3 Input Deck

Click and hold the **MENU** mouse button over the **Commands** menu button to display the SSUPREM3 Commands menu. It will appear as shown in Figure 2-1, showing all the buttons at the top of the frame.

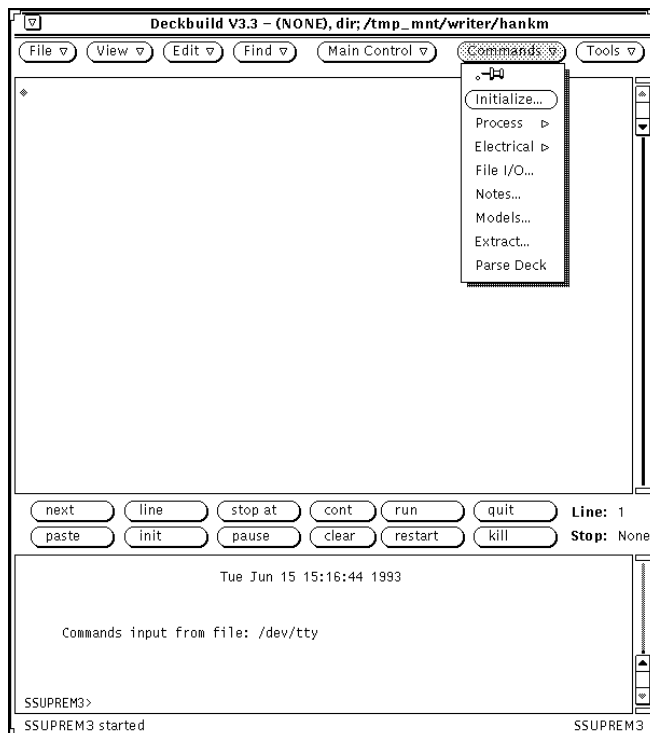


Figure 2-1: Deckbuild SSUPREM3 Commands Menu

Next, move the pointer over the **Initialize...** menu item and release **MENU**. The SSUPREM3 Initialize popup will appear as shown in Figure 2-2.

Figure 2-2: SSUPREM3 Initialize Popup

Choose the material, orientation, thickness, and grid layers by using the pointer and mouse buttons. The default values of these controls have been chosen so that, in many cases, it is unnecessary to change the value of every control on the popup. Just change the values needed. Use lists of items such as **Material** by clicking **MENU** while the pointer is over the square box. A list of materials, resembling a menu, appears. Move the pointer over the desired material and then release **MENU** to activate it.

Change the value of **Thickness** by clicking and holding **SELECT** with the pointer in the small grey slider box, then dragging the box left or right. Release **SELECT** when done. Notice that as the thickness is changed, the grid layers slider automatically follows along to maintain a constant thickness per grid layer ratio. If you not satisfied with the number of layers, change it after the thickness is set. Choose the **Orientation** by clicking **SELECT** over the desired orientation value.

The box containing the value will be indented.

To specify an initial impurity, click **SELECT** in the checkbox to the left of the impurity. The impurity name and slider become active. When the checkboxes are not checked, the impurity information becomes inactive and appears grayed out. Slide the concentration slider to the desired value. Choose an exponent for the concentration by clicking **MENU** over **Exp**. A list of exponents appears. Release **MENU** over the desired value. Finally, write the SSUPREM3 INITIALIZE statement to the deck by clicking on the **WRITE** button.

A line similar to this appears as:

```
INIT SILICON ORIENT=100 THICK=4.00 SPACES=800
```

In this fashion, you can build a process sequence by serially invoking popups from the Commands menu. Most of the commands needed are on the Process pullright menu (Figure 2-3). For example, to open the SSUPREM3 Diffuse popup, click on and hold **MENU** on the **Commands** menu button. The Commands menu appears. While still holding **MENU**, move the pointer down to **Process**.

Note: There is a small, right-pointing triangle to denote a pullright menu. Slide the pointer over to the right a few millimeters, and the Process menu will appear. Move the pointer down to **Diffuse...** and release **MENU** to invoke the popup.

You can pin the Commands, Process, and other menus for convenience. A pinned menu stays in place on the workspace and does not disappear after choosing an item, unlike an unpinned menu. To pin a menu, first invoke it with **MENU**. Then while still holding **MENU**, move the pointer over the pushpin shown at the upper left corner. The pushpin slides into its hole as the pointer moves to the left.

Invoke items from a pinned menu by clicking **SELECT** over the desired item. To unpin a menu, click **SELECT** over the pin. The pin is removed and the menu disappears.

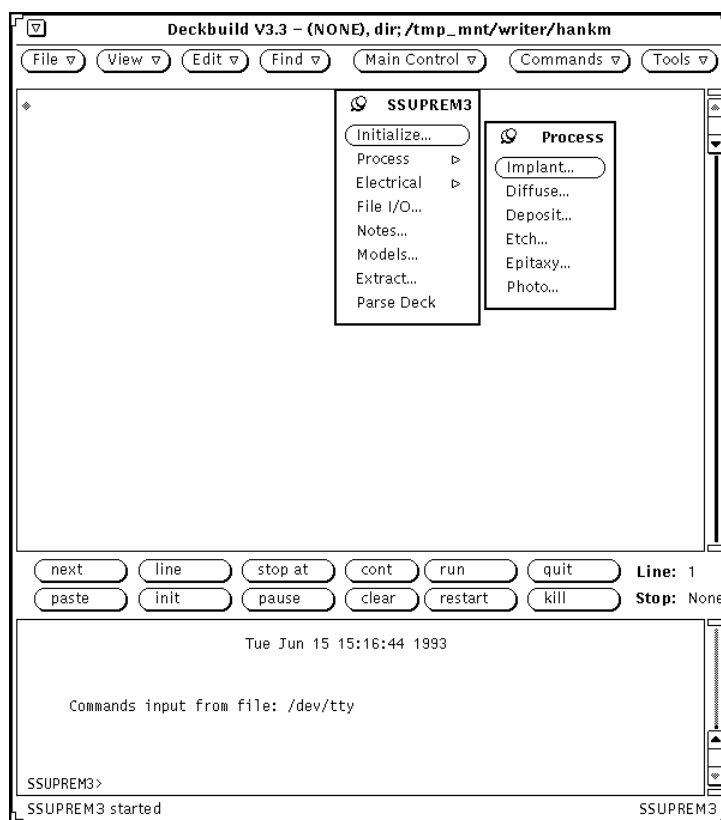


Figure 2-3: Pinned Commands and Process Menus for SSUPREM3

2.1.3: Running A Deck

A SSUPREM3 deck fragment for initial MOS processing is shown in Figure 2-4, which shows this deck in DECKBUILD. Because DECKBUILD was started with the `-s3` option, SSUPREM3 should be running and displaying a prompt in the tty subwindow. If so, you can now run the deck.

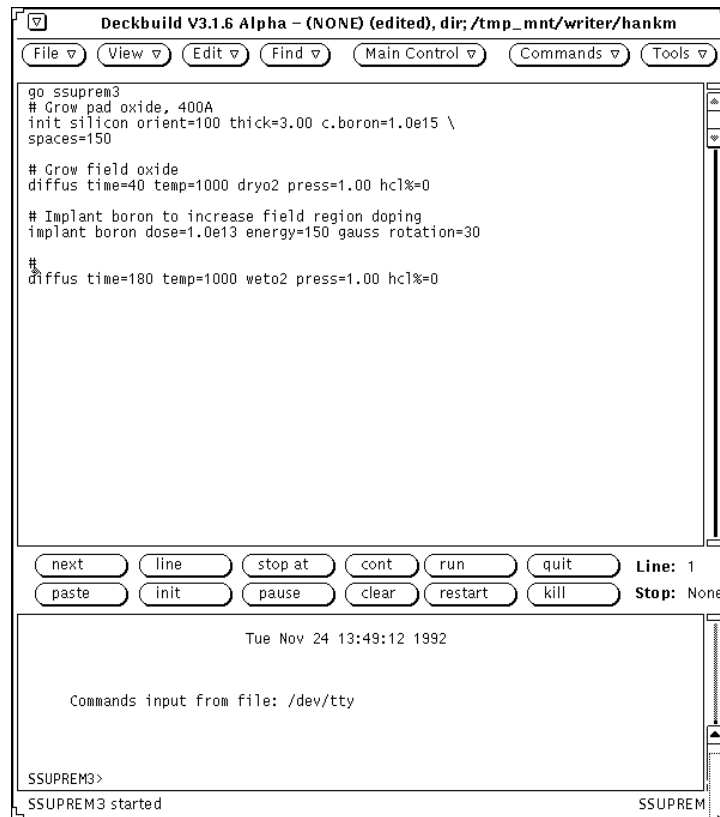


Figure 2-4: SSUPREM3 Deck Fragment

DECKBUILD permits several ways to run the deck: one line at a time, all of the input deck, or only up to a breakpoint. Run the deck a line at a time by clicking **next** on the execution control panel.

The first line (`go ssuprem3`) is ignored by the simulator. This line is an autointerface statement that tells DECKBUILD to run the following deck in SSUPREM3. Click **next** again. The first comment line is executed by SSUPREM3. SSUPREM3 just repeats the comments it sees. Click **next** again to execute the `INIT` statement. Continue in this fashion to go as far as desired into the deck.

At any time, you can tell DECKBUILD to run from the current line to the bottom by clicking **cont**. When **run** is clicked, DECKBUILD always runs from the first line to the last. Be careful if in the middle of the deck, since DECKBUILD starts running from the top again. Use **cont** to keep going from the middle of the deck. DECKBUILD saves special files automatically after each process step. These are history files, which allow backing up and re-starting the simulation from any point in the deck. See Section 3.7: "History" for information about how it works.

Resetting The Current Line

Notice that as the deck is stepped through, the **Line** field is incremented. **Line** shows the current line in the input deck. Since **next** always operates on the current line, it is necessary to reset it when you reach the bottom of the deck if you want to starting from the top again. In fact, the current line can be set to any line in the deck, not just the top line. Set the current line by highlighting part or all of the line in the input deck and click **Line**. The **Line** field will be updated to show the new current line number.

Plotting The Current Structure

DECKBUILD can plot the current device at any point in the process while stepping through the deck. This permits you to visualize what the device looks like at any point. For example, after a gate oxide step in a MOS device, you may want to investigate the oxide thickness, or look at a doping profile after an implant step.

To plot the current structure, first un-highlight any highlighted text on the screen. An easy way to do this is to single-click **SELECT** anywhere in the input deck. Then, click **SELECT** on the **Tools** menu button. DECKBUILD saves the current structure from the simulator then starts TONYPLOT on the structure. In a few moments, TONYPLOT will appear.

Note: Single-clicking **SELECT** on a menu button automatically chooses the default item on the menu, which is **Plot structure**.

TONYPLOT can be kept on the screen for as long as necessary. You can run additional TONYPLOTS later in the process flow to show how the formation of the device progresses. You can also show layout files using the MASKVIEWS layout editor. To start MASKVIEWS, select it from the **Tools** menu.

Saving The Deck

When satisfied with the results of the simulation, save the input deck by choosing **Save as...** from the **File** menu. The **Save As** popup will appear (Figure 2-5).

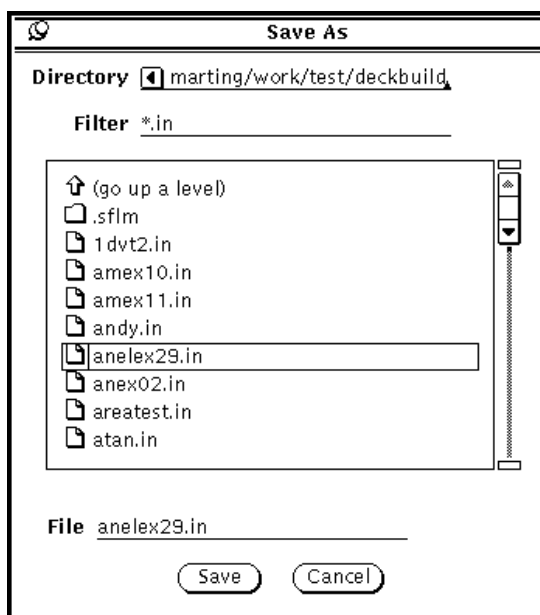


Figure 2-5: Save As Popup

Enter in the directory and the file name wanted and click **Save**. After DECKBUILD saves the file, the popup will disappear.

2.1.4: Quitting DeckBuild

The easiest way to quit DECKBUILD is to select **File→Quit**. This displays a notice to receive confirmation of quitting. Alternatively, you can quit DECKBUILD by using the **Define** menu. The appearance and location of the **Define** menu depends on the window manager in use, but is always available from the header region of the base frame. Under the **Open Look** window manager, click **MENU** with the pointer anywhere in the base frame header. Under the **Motif** window manager, click **SELECT** in the upper left-hand corner of the base frame header. In either case, activate the **Quit** option to quit from DECKBUILD.

If any unsaved edits exist, DECKBUILD displays a notice prompt allowing cancellation of the quit, or discarding of the unsaved edits and quitting. To save the edits, cancel the quit and save the file. You can either save to a new file or, if already editing a file, save edits back to the same file. To do this, select the corresponding option under the **File** menu. See Sections D.1: “Text Subwindow Error Messages” and D.2: “TTY Subwindow Error Messages” for more information.

It is often important to know whether there are unsaved edits, and exactly which file is being edited. DECKBUILD displays the file name in the header bar, or **NONE** if the file is not edited yet. When first saved to a file, it becomes the edited file. The word **edited** follows the file name if there are unsaved edits.

3.1: Starting DeckBuild

3.1.1: Syntax

```
deckbuild [ -s3 | -an | -as | -od | -de | -fa | -ma | -hi |  
-mo | -ut | -ss | -in ]  
[-option [no]exec | [no]auto | [no]commands | [no]write] [-ascii]  
[-simver <simulator_version>][--run][--outfile outfilename]  
[-cutfile <cutfilename>] [--optfile <optfilename>]  
[-remote <hostname>][--editfont <font>]  
[-inpipe <input pipe> --outpipe <output pipe>] [--noplot] [--help]  
[xview-arguments] [textedit-arguments] filename
```

3.1.2: Description

You can start DECKBUILD in either an interactive mode or a batch mode. In the interactive mode, it is possible to create, edit, and run input decks using mouse and keyboard operations. In the batch mode, DECKBUILD runs a previously created input deck. In the interactive mode, DECKBUILD appears as a window containing a text subwindow and a tty subwindow. The filename specifies the file to edit. If not specified, the text subwindow will be empty. If specified, DECKBUILD loads the file into the text subwindow.

In batch mode, a filename is required. DECKBUILD appears as an icon and automatically starts a simulator and executes the entire input deck (see “Default Simulator” on page 3-3). DECKBUILD quits when the run is complete. In either mode, you can save the run-time output of the simulation by specifying the `--outfile` option.

3.1.3: Options

The following options choose the default simulator configuration. A detailed explanation of these options is provided “Default Simulator” on page 3-3.

```
-s3 ssuprem3 -an athena  
-od clever -as atlas -ma masksim  
-de devedit -ut utmost -mo mocasim  
-ss smartspice -in internal -hi hipex
```

- **-run** starts DECKBUILD in batch mode. The input deck filename is required. If none is specified, DECKBUILD displays an error message and exits.
- **-outfile <outfilename>**— The file specified by outfilename is created to store the run-time output of the simulation. DECKBUILD writes each line of the tty subwindow to outfilename as the simulation progresses.
- **-cutfile <cutfilename>** — The file specified by cutfilename is loaded as the current cutline file. DECKBUILD uses the file for mask, region, and electrode data.
- **-optfile <optfilename>** can be used to load an optimizer data file with the input deck. The input deck must be specified and must match the optimizer data file. You can also use it with `--run` (synonym: `-opt`) to submit batch optimization runs.

- **xview/textedit-arguments** — DECKBUILD accepts standard OPEN LOOK and textedit command-line arguments. XView arguments can set such things as the color, font, and layout of DECKBUILD, while textedit arguments control such things as history limit, margins, and tab spacing.
- **filename** specifies the name of the input deck that DECKBUILD should initially load. If no filename is specified, DECKBUILD leaves the text subwindow empty. If the specified file does not exist, DECKBUILD displays a notice prompt to confirm its creation.
- **-option [no]exec | [no]auto | [no]commands | [no]write** toggles these **Main Control** options for a particular run. The options are `exec` for execute simulator, `auto` for auto interface, `commands` for the Commands menu, and `write` for write text to text window (all corresponding to the options setting on the Main Control popup). A 'no' indicates to turn the option off. Repeat the `-option` argument to enable/disable more than one option.
- **-ascii** enables DECKBUILD to run in a non X windows environment. No popups or windows are created, but an input deck can be run normally. This option requires the use of an input filename and the `-run` option. The `-outfile` option (to store run-time output) is also helpful. If `-outfile` is not specified, the run-time output goes to `stdout`.
- **-simver <simulator_version>** specifies that the simulator should be started with the specified version. The simulator are invoked as `<simulator> -V <simulator_version>`. If no simulator is specified, the default simulator uses this version.

Note: `-sv` is also accepted as well as `-simver`.

- **-help** displays a list of the DECKBUILD and XView command line options.
- **-remote <hostname>** sets current simulations to be executed on the specified host. Although this option for interactive use can be used, it is intended for batch mode simulation. In interactive mode, you can select remote hosts from the **Simulator Properties** popup. See Section 3.13: "Remote Simulation" for more information.
- **-editfont <fontname>** allows the font for the text and tty sub windows to be specified as required.
- **-noplot** specifies that no plot commands within the deck are executed.
- **-inpipe <input pipe> -outpipe <output pipe>** specifies input and output pipe for communication with other products.

Examples

The following command will start DECKBUILD in interactive mode and pre-load the specified file.

```
deckbuild [filename.in] &
```

DECKBUILD can be submitted as a batch command on the UNIX command line. This method runs an input deck and quits at the end of the deck. You can submit a number of jobs for serial execution in this manner. The format of the command uses the `-run` option as follows:

```
deckbuild -an -run [filename.in]
```

DECKBUILD appears on the screen as a closed Icon and execute the named input deck, it may be opened to a full screen at any time during the execution. DECKBUILD exits completely when the last command in the input deck has been executed. If the runtime output is required to be stored into a separate file, the following options can be used:

```
deckbuild -an -run [filename.in] -outfile [filename]
```

Again, DECKBUILD appears as an Icon and executes the specified input deck. But in this case, all runtime output is appended to the named outfile.

If simulations are executed while X Windows is not running (or in a screen locked mode), the `-ascii` option can be used as follows:

```
deckbuild -an -run [filename.in] -outfile [filename] -ascii
```

DECKBUILD does appear as an Icon and executes the specified input deck. All runtime output is again appended to the named outfile, as recommended for `-ascii` use. If no outfile is specified for this type of command the runtime output is displayed in the current command tool.

Structure files saved during batch jobs should be carefully thought out. Make sure not to over write structure files with subsequent runs in the same working directory.

Defaults

A large number of control settings and options are configured at startup time. To configure these default options, use the **Save** function in the DECKBUILD property popups. DECKBUILD provides defaults saving routines on its property popups.

Default Simulator

At startup, DECKBUILD creates popups for the chosen default simulator and starts the default simulator in the tty subwindow if you enable the tty subwindow. DECKBUILD determines the default simulator according to the following rules:

1. The simulator specified on the command line, if any, takes precedence over all other rules.
2. If no simulator was specified on the command line, DECKBUILD uses the last saved default simulator. The default simulator is saved by clicking the **Save as Defaults** button on the Control Pad.
3. If no default simulator has been saved, DECKBUILD uses ATHENA.

Note: DeckBuild automatically changes simulators whenever it encounters an `autointerface` statement in the input deck (see “Autointerface” on page 1-2). Command line specification of the default simulator is important in the batch mode if there is no initial `autointerface` statement in the input deck.

3.2: DeckBuild Controls

DECKBUILD consists of a window (Figure 3-1) containing two subwindows: the text subwindow in the upper half of the base frame and the tty subwindow in the lower half. The text subwindow is used to build and edit input decks, while the tty subwindow is used to run the simulation. You can also display a messages subwindow below the tty subwindow.

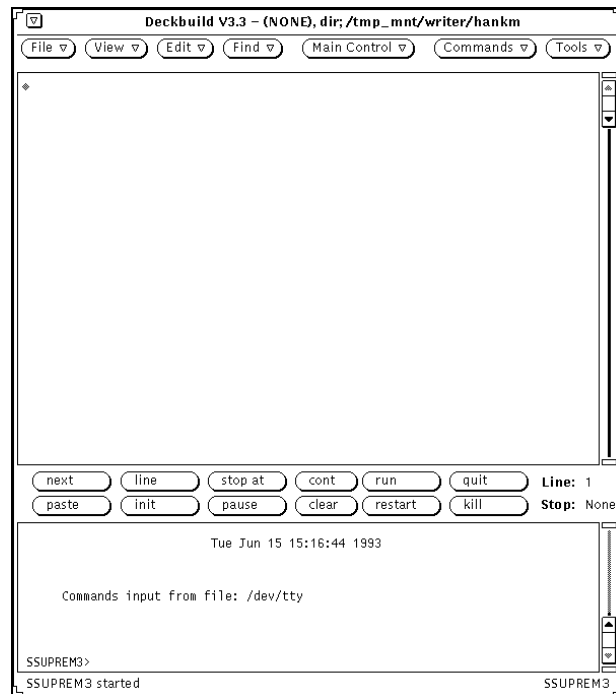


Figure 3-1: DeckBuild Base Window

3.2.1: Main Window Layout

DECKBUILD's controls are laid out in a hierarchical format. The most important and frequently-used controls are placed on the main window. Lower-level, less frequently used controls are stored as menu items on the top-level menus, or on popup windows accessed from the top-level windows. The resulting design attains an optimum of flexibility, ease of use, with a minimum of wasted space on the screen.

Menu Buttons

The set of buttons located along the top of the main window are the menu buttons. These buttons allow access to file control, simulator selection, simulator commands, and the tool palette and are as follows:

- **File** — The pulldown menu for saving and retrieving input deck files.
- **View** — The pulldown menu for changing the view of the text.
- **Edit** — The pulldown menu for cut, paste, and undo editing operations.
- **Find** — The pulldown menu for text search and replace operations.
- **Main Control** — The pulldown menu for top-level DECKBUILD configuration and control.
- **Commands** — The pulldown menu for simulator-specific commands and input deck creation.
- **Tools** — The pulldown menu for invoking the VWF INTERACTIVE TOOLS such as TONYPLOT.

Execution Control Buttons

The set of buttons located between the text and tty subwindows are the execution control buttons. These buttons allow complete interactive runtime control of the simulator and are as follows:

- **next** — This sends the current line to the simulator, and advances the current line by one. If a simulator is not running, one will be started.
- **line** — This resets the current line to the selected line.
- **stop at line** — This sets the breakpoint to the currently selected line.
- **stop now** — This stops execution after completing the current command and the associated history file's save command, if appropriate.
- **cont** — This continues the simulation from the current line to the end of deck, or to the breakpoint, if any. If not running, it also starts the simulator.
- **run** — This runs deck from the top to the bottom, or to the breakpoint, if any. If not running, it also starts the simulator.
- **quit** — This sends a quit statement to the simulator.
- **paste** — This sends the current selection to the simulator to be executed.
- **init** — If the selected text is a file, then the correct simulator is initialized with the file. Otherwise, the selected line is used to initialize from history.
- **pause/unpause** — This pauses/unpauses the running simulator.
- **clear** — This unsets the current breakpoint.
- **restart** — This restarts the current simulator if it's not running.
- **kill** — This kills the simulator.

3.2.2: Text & TTY Subwindows

Although the primary means of building and modifying input decks in DECKBUILD is via popup windows, DECKBUILD's text subwindow supports a full-featured text editor that allows editing decks directly. It is not necessary to know any special editing commands because they are available from the main window header menus. Command menus are accessed via the File, View, Edit, and Find menu buttons, and also by clicking MENU with the pointer anywhere in the text subwindow.

3.2.3: Using the Text Subwindow

Creating A New File

To store a new file, move the pointer to the File menu button and click and hold the **MENU** mouse button. Move the pointer to the **Save as...** menu item and release the **MENU** button. A **Save As** popup are displayed showing a list of directories and files in the current working directory (Figure 3-2).

To move between directories, either modify the **Directory** field and press RETURN or double-click on the required directory in the list. To save the required file, either highlight it in the list and select the **Save** button, double-click on the required file or enter a new file name in the **File** field and select the **Save** button.

Note: Saving a file to a new directory, moves DECKBUILD's current working directory to that location.

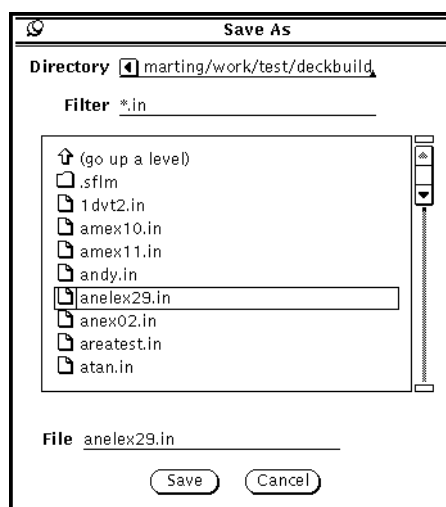


Figure 3-2: Save As Popup

Loading an Existing File

Any plain ASCII text file can be loaded into DECKBUILD. To load an existing file, choose the **Open...** item from the **File** menu. A file loader popup is displayed showing a list of directories and files, corresponding to the **Filter** field, in the current working directory (Figure 3-3). To move between directories, either modify the **Directory** field and press the Return key or double-click on the required directory in the list. To load the required file, either highlight it in the list and select the **Open** button or double-click on the required file.

Note: Loading a file from a new directory, moves DECKBUILD's current working directory to that location

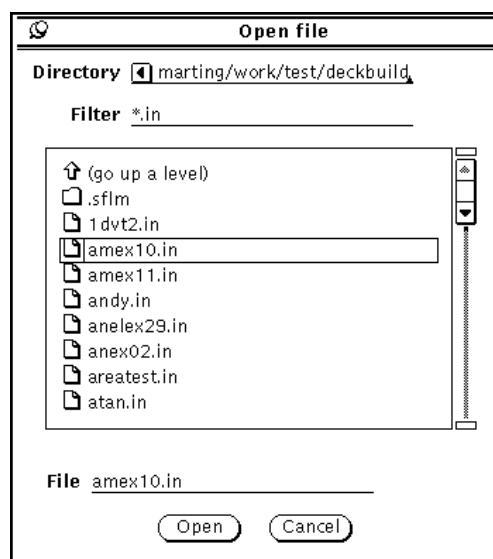


Figure 3-3: Open File Popup

Saving Changes

After changing the contents of an edited file (for example, by changing a diffusion parameter or by appending a device test to the end of a process deck), you can save the changes back to the same file or save the deck as a new file. To save the changes back to the same file, choose **Save** from the **File** menu. DECKBUILD saves the current file and makes a backup of the original renamed with a % suffix. For example, a file called `nmos.in` would be backed up to a file called `nmos.in%`. To save the changes to a new file, choose **Save as...** from the **File** menu. The procedure is the same as described above for saving a newly-created file.

Editing Input Decks

DECKBUILD allows you to search for text strings in a deck, copy text from one location and paste it to another, and add/delete text. To search for a text string, choose **Find and Replace** from the **Find** menu. The **Find and Replace** popup window appears with fields for text strings to find and to replace. Enter the text string to find on the first line. If desired, also type in the replacement text on the next line.

To search for each occurrence of a string, click **SELECT** on the **Find** button. Successive clicks find successive occurrences of the text. To replace a text string and approve each occurrence, click on the **Find** button to find each string. Then, click **Replace** to replace the text.

At each instance of the text string, if it is desired to replace the text and then search again, click **Replace** then **Find**.

To change only the first instance of a string without approval, click **Find** then **Replace**. To replace all instances of the text string immediately, choose **Replace All**.

Adding And Deleting Text

Add text to the deck by placing the insert point anywhere in the text subwindow (by clicking **SELECT** on the desired location) and click the **WRITE** button from any simulator pop-up window, or by simply entering the text. Delete text by choosing the insert point, then backspace over the text to delete.

Another way to insert and delete text is by using the system clipboard. To cut (that is, delete) text from the deck, click **SELECT** on the first character to cut. Then, click **ADJUST** on the last character to cut. This operation selects the range of text to cut. Finally, choose **Cut** from the **Edit** menu. The text is deleted from the input deck, and is placed in the system clipboard.

To paste (insert) text from the clipboard, place the insert point at the desired location, then choose **Paste** from the **Edit** menu. The text from the clipboard appears at the new location.

Copying Text

To copy a section of text, select the text to be copied (using one of the select mechanisms described above), then choose **Copy** from the **Edit** menu. This copies the selected text to the clipboard. To insert the text in the document, place the insert point at the desired location and choose **Paste**.

Tips On Cutting And Pasting

The following provides useful tips for cutting and pasting when working in the text Subwindow:

- Single-clicking **SELECT** in text selects a single character, double-clicking selects a word, triple-clicking selects a line, and clicking four times selects the entire document.
- Some may prefer the **Cut**, **Paste**, and **Copy** buttons on the keyboard (not available on all keyboards) rather than using the **Edit** menu.
- Text copied to the clipboard remains there until another copy operation is done.
- Text may be copied to the clipboard from one application, and pasted into an entirely different application.
- DECKBUILD's tty subwindow also supports cut and paste operations.

3.2.4: Using the TTY Subwindow

The tty subwindow, used to drive the simulation programs, is a text-based command window. It accepts many of the same commands and has many of the same capabilities as the text editor.

Entering Commands Directly

The execution control buttons provide the primary means of input to the running simulator, but text typed directly into the tty subwindow can be used as well. Place the pointer anywhere in the tty subwindow and click on **SELECT**. If a simulator is running and is waiting for input, a caret appears at the simulator prompt. Commands typed at the keyboard are executed by the simulator.

The text of the current command line can be edited using the normal text editing functions explained above.

Editing The Contents

Except for the current command line, all lines in the tty subwindow are read-only and cannot be edited. The subwindow displays a log of the simulator input and output, which can be scrolled using the scrollbar. To search for a particular text string, use the **Find** and **Replace** popup. Bring up the tty (Term Pane) menu by placing the pointer anywhere in the tty subwindow and click on **MENU**. This menu contains a number of items used to edit and manipulate the tty subwindow, and is like the text menu. Choose **Find and Replace** from under the **Find** menu item. Searching is done the same as with the text subwindow.

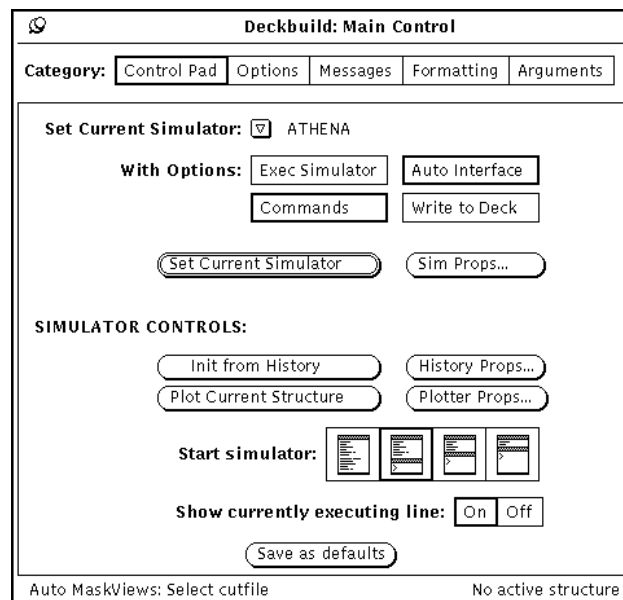


Figure 3-4: Main Control Popup

Cut, Paste, And Copy

You can copy and paste text in the tty subwindow directly by using the **Edit** menu from the tty menu or by using the corresponding keys if available. Cutting is not allowed on any but the current command line, however, since the remainder of the subwindow contents are read-only.

Saving/Resetting The Contents

To reset the contents of the **tty** subwindow (that is, remove the log and clear the subwindow), bring up the **tty** menu and choose **History→Clear log** (Figure 3-4). To save the contents of the **tty** subwindow, bring up the **tty** menu and choose **History→Store log as new file**.

Note: **History** on this menu refers only to the contents of the **tty** subwindow, and is not related to DECKBUILD's **History** feature.

Error Messages

The Error Messages section at the end of this chapter contains information about error messages that may be encountered while using the text and **tty** subwindows.

3.3: Main Control

The **Main Control** popup (Figure 3-4) provides a collection of controls, organized by category, that allows customizing of DECKBUILD's configuration.

The **Main Control** popup with the **Category** menu displayed is shown in Figure 3-3. The **Category** menu contains the following control category selections.

- **Control Pad** — Simulator choice, activation, and runtime options.
- **Options** — User configurable option settings.
- **Messages** — DECKBUILD debugging control.
- **Formatting** — Input deck operation formatting.
- **Arguments** — Command line arguments for other VWF INTERACTIVE TOOLS.

Choose a category by clicking on **MENU** with the pointer over **Category**. Move the pointer over the category of interest, then release the **MENU** button. When you select a new category, the popup changes from the old category to the new. Old controls disappear, new ones appear, and the popup may change its size.

Each category contains a **Save as defaults** button. Click on the button to save the settings in that category. DECKBUILD configures itself with the saved settings the next time it is invoked.

3.3.1: Control Pad

The **Control Pad** provides one-stop service for the highest level of DECKBUILD configuration and runtime control. Choose the currently configured simulator, auto interface options, and DECKBUILD layout using the **Control Pad**. In addition, the important runtime control features of **History** and **Plot** are placed here for easy and quick use.

3.3.2: Choosing a Simulator

Choose a new simulator when writing or modifying a deck for that simulator, to see its **Commands** menu, or to shut down the currently running simulator and start up the new simulator. DECKBUILD provides control over each of these actions with **Options**. To change over to a new simulator, click and hold **MENU** in the **Set Current Simulator** setting. A list of available simulators is shown (Figure 3-5). Move the pointer over the desired simulator and release the **MENU** button. Activate the options that are appropriate (usually all except **Write to Deck**). Finally, click on the **Set Current Simulator** button.

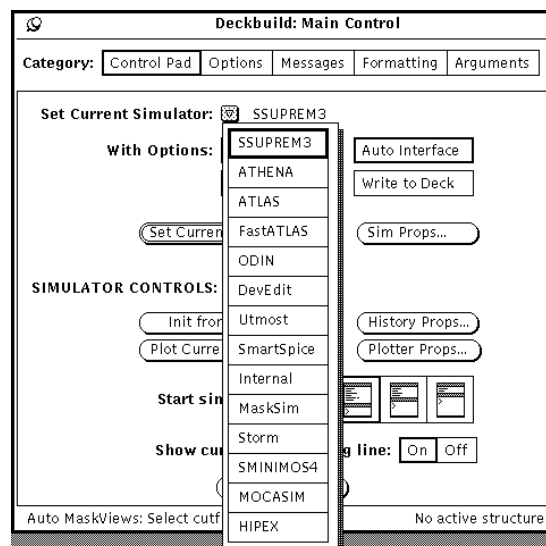


Figure 3-5: Main Control - Set Current Simulator

In the following section, the current simulator means the simulator before the button is clicked, and the new simulator means the current simulator after the button is clicked. Based on the chosen options, the following occurs:

1. **Exec Simulator** causes the currently running simulator to shut down and the new simulator to start. If you enable the **Auto Interface** option and DECKBUILD determines that an interface is appropriate, then DECKBUILD performs an auto interface from the current to new simulator. **Exec Simulator** is enabled by default.
2. **Auto Interface** causes an auto interface to be performed between the current and new simulator. This option is also referenced at run time whenever DECKBUILD encounters an auto interface statement in the input deck. See Section 3.8: “Auto Interfacing” for more information. **Auto Interface** is enabled by default.
3. **Commands** causes DECKBUILD to change the **Commands** pull-down menu to reflect the new simulator’s syntax. All popups for the current simulator that are not pinned are closed. **Commands** is enabled by default.
4. **Write to Deck** causes an auto interface statement to be inserted in the input deck at the location of the text caret. See Section 4.3: “AUTOELECTRODE” for more information. **Write to Deck** is not enabled by default.

Note: DECKBUILD always delays popup creation until the popups are needed to reduce startup time. The **Commands** menu and associated popup windows are created only when referenced for the first time. Therefore, if the new simulator has not been referenced before, then there is a short delay while the popups are created. Otherwise, the change will be instantaneous. A notice in the lower left footer of the main window will be displayed while the popups are being created.

When the change from one simulator to another is complete, the new current simulator is shown in the lower right footer of the base window. The current simulator is always shown in the lower right footer.

3.3.3: Simulator Properties

To access properties unique to each simulator, click on the **Sim Props** button. A popup is displayed that contains settings and options applicable to the current simulator, including a switch for local and remote Simulation and command-line arguments used to execute the simulator. If remote simulation is selected, the hostname of the machine that executes the simulator must be specified. See Section 3.13: “Remote Simulation” for more information.

3.3.4: Start Simulator

The **Start Simulator** setting determines DECKBUILD’s window layout. Each of the four choices in the setting has an iconical representation of what DECKBUILD can look like: text subwindow only, or text subwindow with small, medium, or large tty subwindow displayed. By default, DECKBUILD appears with the tty subwindow displayed in the small configuration.

The current simulator is started when the tty subwindow is enabled. From that point on, the simulator continues to run, even if the tty subwindow is made to disappear.

3.3.5: Simulator Controls

Three high-level run-time functions are provided on the **Control Pad: Init from History, Plot Current Structure, and Show Currently Executing Line.**

Init from History is used to re-initialize the simulator from some previously-run line in the input deck. DECKBUILD automatically saves files as each line in the deck is run. This permits transparent movement back and forth in the input deck if you need to back up to try something again. **History** properties, including the ability to disable history, are accessed by clicking on the **History Props...** button. See Section 3.7: “History” for more information.

Plot Current Structure is used to plot either the current structure or any selected structure. It provides a shortcut to **Plot** on the **Tools** menu. The following occurs when you click on this button.

- If there is text selected (highlighted) anywhere on the screen, DECKBUILD takes the text as filename of a file to plot. DECKBUILD starts up TONYPLOT on the named file.
- If no text is selected and a simulator is running, DECKBUILD causes the simulator to save its simulation data and then starts TONYPLOT using that data. This does not disrupt any lines of input deck waiting to be executed by the simulator.
- Optionally, choose a set file by clicking on the **Plotter Props...** button. The set file is used to record a given plot's layout, such as scaling, zoom, number and type of plots shown. After creating a set file, you can use it to re-create the same layout when using the same or any other plot data. A set file is often useful for comparing the results of different simulation runs.

After pressing the **Plotter Props...** button, the **Plotter Set Files** popup will appear. This contains a scrolling list of set files in the current directory, and a text fields used to search for set files. Adjust the directory name and directory filter if necessary. Click **SELECT** over the name of the desired set file, if any, in the scrolling list. If none are desired, then make sure no entries are selected (de-select a selected list entry by clicking **SELECT** on it again). The selected entry, if any, will be used as the set file on subsequent plots.

It is also possible to specify when DECKBUILD should save the active structure (no filename highlighted). See the **Plot structure** description in Section 3.3.6: "Options Category". It provides a shortcut to **Plot** on the **Tools** menu.

Show Currently Executing Line tells DECKBUILD whether or not to select (highlight) each line in the input deck as it is executed by the simulator. DECKBUILD also automatically scrolls the text subwindow to keep the currently executing line always in view. This feature is enabled by default, except when running batch mode (`-run`).

3.3.6: Options Category

The **Options** category (Figure 3-6) is a collection of a number of settings that modify the behavior of various DECKBUILD functions. For example, it is possible to configure what happens when **WRITE** is clicked on a syntax popup, whether history files should be automatically removed at the end of a run, and even the nice value of the simulator.

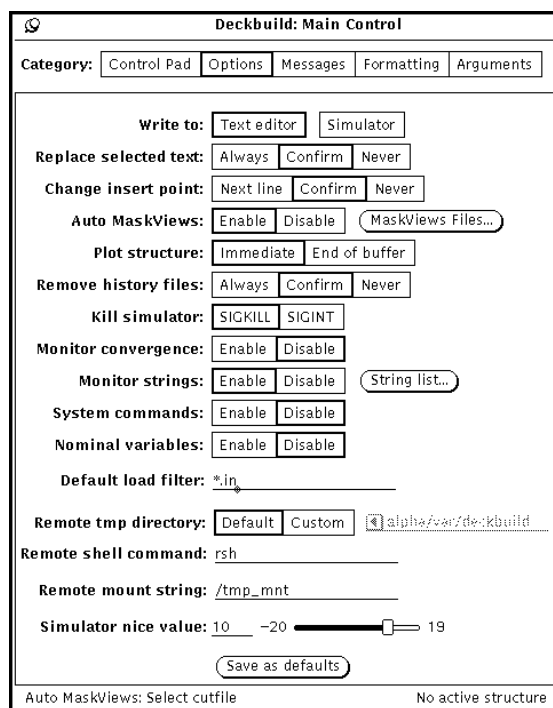


Figure 3-6: Main Control - Options Category

Write to setting determines what happens when **WRITE** is clicked on any of the syntax popups. It allows choosing to send the text to either the text subwindow or directly to the simulator running in the tty subwindow or both. The default is **Text Editor** only.

Replace selected text tells DECKBUILD what to do when **WRITE** is clicked on while any line of text is selected (highlighted) in the text subwindow. This situation often comes about after parsing a line of the deck, see Section 3.5.2: “Parsing the Deck”. DECKBUILD provides the choice of replacing the selected text with no confirmation, confirming each replacement, or not allowing replacement at all. If you activate **Confirm**, then DECKBUILD displays a notice prompt to either confirm or cancel the overwrite each time. If you activate **Never**, DECKBUILD displays a notice prompt confirming that the operation was cancelled. The default is **Confirm**.

Change insert point modifies the **WRITE** insertion behavior. This feature is designed to make deck building easier by checking the location of the text caret each time **WRITE** is clicked on. The caret should be at the beginning of a line. It could be relocated for various editing purposes, resulting in the cursor being left in the middle of a line. **Next Line** automatically moves the caret to the beginning of the next line. **Confirm** asks for confirmation or cancellation of the move each time. If confirmed, DECKBUILD moves the caret to the next line. **Never** inserts text wherever the cursor is located. The default is **Confirm**.

Auto MaskViews enables or disables the automatic substitution of MASKVIEWS layout information during deck execution. DECKBUILD also displays the status of this choice in the left footer of the **Main Control** popup for easy reference during run time. The **MaskViews Files...** button is placed here as a convenient way to select cut files (also accessible from the **Cut files...** under MASKVIEWS on the Tools

menu). See Section 4.10: “MASKVIEWS” for a complete description on how to use it with DECKBUILD. The default is **Enable**.

Plot structure controls where interactive plots are made. Interactive plots are done from either the **Tools** menu or from the **Control Pad**. Normally, DECKBUILD saves and plots the active structure right away. The simulator, however, may be busy executing several lines from the input deck. This can happen, for example, after clicking on the **run** button, which causes DECKBUILD to queue up many lines from the deck to be run and feeds them down one at a time. In this case, it is possible to define whether the save and plot commands are placed at the beginning or at the end of those queued commands. Choose either **Immediate** or **End of Buffer**. The default is **Immediate**.

Remove history files defines what DECKBUILD does with history files when DECKBUILD exits. Choosing **Always** causes DECKBUILD to always clean up history files after itself. **Confirm** brings up a confirmation notice prompt when DECKBUILD exits. You can then confirm or cancel history file removal. **Never** ignores history files and does not elicit a notice prompt. This is useful if you want to modify the history files on a regular basis.

Note: It's OK when history files are not removed as long as there's enough disk space. They'll eventually start getting re-used, so they won't pile up endlessly. See Section 3.7: “History” for more information. The default is **Confirm**.

Kill simulator determines how DECKBUILD kills the simulator when the **kill** button is clicked on. SIGKILL is guaranteed to kill the simulator, but it does not give the simulator a chance to clean up after itself. If the simulator uses any temporary files when killed, the temporary files will not be removed and will remain in the current directory, or in /tmp. It uses the Unix “kill” signal 9. On the other hand, SIGINT allows the simulator to remove its temporary files and perform any other required actions before it terminates. It uses the Unix “interrupt” signal 2. We recommend that you use SIGINT for normal use. If the simulator does not seem to die properly under some circumstances, switch to the forced kill of SIGKILL.

Monitor convergence, if enabled, monitors the output from ATLAS and looks for messages indicating that the simulator has failed to converge. If convergence failure is detected, an error message is displayed and the simulation is halted. No more lines from the input deck are sent to the simulator, and the simulator is left running and displaying its interactive prompt. You should enable this option to stop ATLAS runs at the first sign of convergence failure. You should disable this option for doing snapback characteristic simulation, which actually depends on convergence failure as a precondition to switching boundary conditions and continuing the simulation.

Monitor Strings, if enabled, monitors the TTY output for strings selected in the **Monitor String List** (Figure 3-7). If a selected string is detected, a message is displayed and the current simulation is stopped at this point. Note the simulator stays active.

System Commands, if enabled, allows DECKBUILD to execute UNIX system commands within a simulation deck. To use a system command, the line must start with the command system as shown below.

```
system rm.history*.str
```

To enable system commands for VWF AUTOMATION TOOLS, set the environment variable DB_SYSTEM_OPTION to any value.

Nominal Variables, if enabled, allows you to use the nominal flag for DECKBUILD set variables. If you specify the nominal flag in a set statement, the variable remains unchanged if already in existence. The variable will be created as specified if it is new. See Section 4.11: “SET” for more details on the set command.

Default Load Filter sets the default file filter for DECKBUILD's file loader popup.

Remote options are used to perform remote simulation. These options are explained in Section 3.13: “Remote Simulation”.

Simulator nice value sets the priority of the simulator process. Negative nice values give the simulator process more CPU time relative to other processes. Positive nice values give it less. Only the super-user can use negative values. The value is used only when the simulator is started and does not change the nice value of the simulator once it is running. You must then quit and restart the simulator to give it a new nice value. The default value is 10 (added).

To set the simulator nice value for VWF AUTOMATION TOOLS, set the environment variable NICE_ARG to the required number.

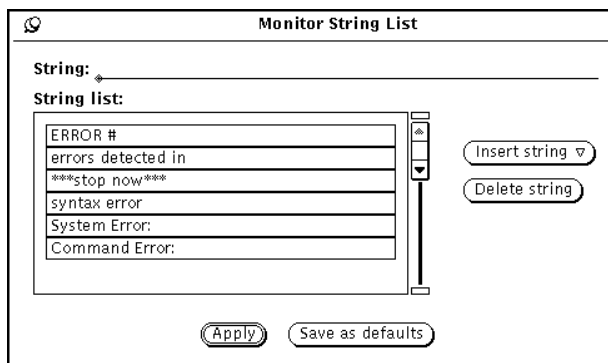


Figure 3-7: Monitor String List Popup

3.3.7: Messages Category

The **Messages** category (Figure 3-8) contains option settings that control the display and behavior of DECKBUILD’s messages window.

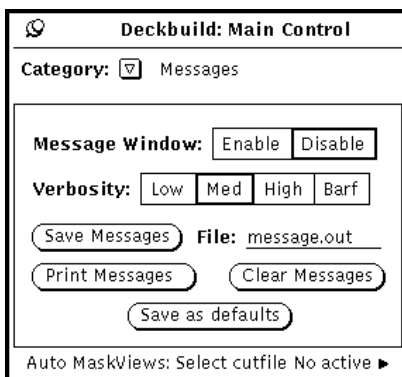


Figure 3-8: Main Control - Messages Category

The **Message** Window provides a debugging log of various actions that DECKBUILD takes as it runs, such as input decks and switches between simulators. The **Message** Window is useful to track user activity in the input deck since activity messages are all time-stamped. This times how long various sections of the input deck take to execute.

The window is enabled with the **Message Window** setting. When enabled, the window appears as a small, read-only text window at the bottom of the main window, and below the tty subwindow.

Messages are logged as a function of the selected **Verbosity** level. A general rule of thumb is as follows:

- **Low** — Tracks major events, such as simulator starts, stops, and simulator switching. Also, many warning messages that appear on the screen are logged at this level.

- **Med** — Low plus auto interfacing information. This is the default setting.
- **High** — Med plus input deck lines as they are queued to be run.
- **Barf** — High plus input deck lines as they are actually executed.

3.3.8: Formatting Category

The **Formatting** category (Figure 3-9) controls the format and initial state of input deck operations, and is used in conjunction with the VWF.

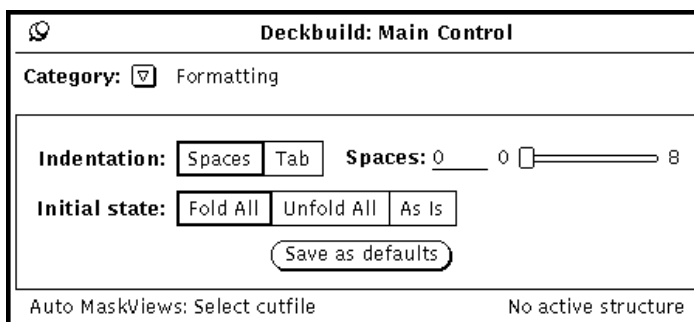


Figure 3-9: Formatting Category

Indentation sets the number of spaces that each level of operations is indented. Top-level operations are flush with the left margin, while operations they contain are indented by this amount and so on with each level of nested operations.

Initial state controls what the format of the deck will be when DECKBUILD is invoked from the VWF. Operations can either be completely folded so that only top-level operations appear, or completely unfolded so that all operations appear in their entirety. The deck can also be left as-is in the state when it was saved to the database.

3.3.9: Arguments Category

The **Arguments** category (Figure 3-10) sets the default command-line arguments used by DECKBUILD to start certain other VWF INTERACTIVE TOOLS.

To change command-line arguments, or to use a different program, enter the new arguments and click on the Return key. The new arguments are used the next time that program is started.

DECKBUILD automatically appends specific arguments to the ones that were entered. For example, DECKBUILD appends a filename to the **Plotter** argument whenever the current structure is plotted, and appends a layout filename when MASKVIEWS is started. For this reason, it is unnecessary (or wise) to put specific filenames in the argument lists.

When encountering a `tonyplot` statement during execution, DECKBUILD also automatically determines if a structure is 3D and use the appropriate **3D plotter** argument.

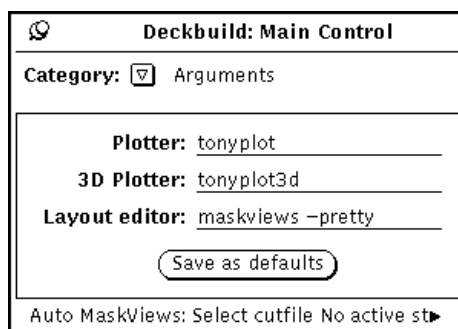


Figure 3-10: Main Control Arguments

3.4: Execution Control

The execution control buttons, grouped between the text and tty subwindows when the tty subwindow is enabled, control how the input deck is sent to the simulator. It is possible to step through the deck a line at a time, run up to a point, run the entire deck, or even back up and continue. DECKBUILD provides the features needed to start, run, and quit the simulation.

3.4.1: Execution Concepts

DECKBUILD always remembers the current line in the input deck. The current line is the line that is next sent to the simulator, and is always shown next to **Line** on the right side of the control panel. It is updated automatically when stepping through or running the input deck, initialize from a filename, re-initialize from history, or when explicitly reset.

DECKBUILD sends lines one at a time to the simulator when the simulator is ready for more input. Lines that have been sent to the simulator (using the **next** or **run** buttons, for example) are buffered until the simulator is ready to accept them. Therefore, you can send down an arbitrary number of lines to be simulated all in one go, and the lines will be executed in the order received. The buffer is cleared automatically whenever the simulator exits. See Figure 3-11 for a view of the **Execution Control** buttons.

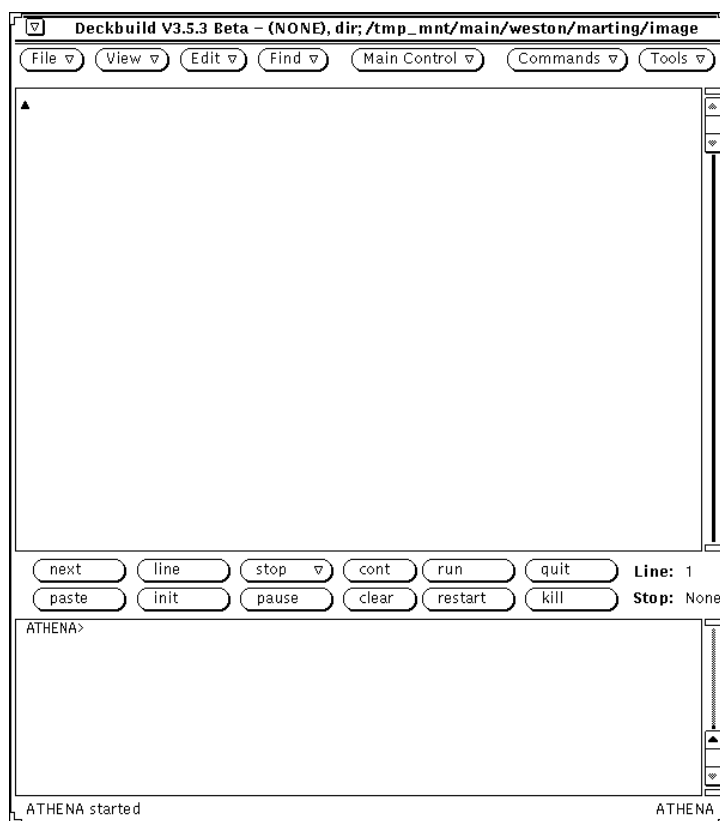


Figure 3-11: The Execution Control Buttons

3.4.2: Execution Control Buttons

The following describes the execution control buttons.

- **next** — Sends the current line to the simulator and advances the current line by one.
- **line** — Resets the current line to the currently selected line.
- **stop at line** — Sets the breakpoint to the currently selected line.
- **stop now** — Stops current execution after completing the current command and the associated history file's save command, if appropriate.
- **cont** — Continues the simulation from the current line to the end of deck or to the breakpoint, if any.
- **run** — Runs the deck from the first to the last line or to the breakpoint, if any.
- **quit** — Sends a quit statement to the simulator.
- **paste** — Sends the current selection to the simulator to be executed. The current selection may exist in any application, or in DECKBUILD itself. Use **paste**, for example, to paste in lines from another input deck that might have been viewed using a text editor.
- **init** — If the selected text is a file, then the correct simulator is initialized with the file. Otherwise, the selected line is used to initialize from history.
- **pause/unpause** — Pauses/Unpauses the simulator.
- **clear** — Unsets the current breakpoint.
- **restart** — Restarts the current simulator if it is not running.
- **kill** — Kills the running simulator.

3.4.3: Stepping Through and Running the Deck

You can execute one line at a time by pressing the **next** button. The current line is sent to the simulator, and the text caret moves to the next line. Press **next** again to execute the next line. Continue stepping lines through all or part of the deck.

Alternately, run the whole deck by pressing **run**. The deck is executed from the first line to the last. Press **cont** to continue onwards from the current line all the way to the end of the deck. Both **run** and **cont** stop at the breakpoint if one is set.

You can stop the execution at any point by pressing the **now** option on the **stop** menu. This does not quit the simulator, but halts the execution after completing the current command, along with associated history file save if appropriate.

3.4.4: Setting and Clearing Breakpoints

If you want to run the input deck only up to a certain line, set a breakpoint on that line. DECKBUILD runs up to, but not including, the breakpoint. Set the breakpoint by selecting (highlighting) all or any part of the desired line, then click on the **at line** option on the **stop** menu. The breakpoint is displayed as **Stop** on the right side of the execution panel.

Clear the breakpoint by clicking on the **clear** button.

If multiple breakpoints are required in a deck, the use of the **Monitor Strings** option stops the execution at locations marked by selected comments. For example, if the comment `# stop here` were entered at different positions in an input deck and this string was then added to the enabled **Monitor Strings** list, the simulation would stop at each location with a message.

3.4.5: Setting the Current Line

As already mentioned, the current line is automatically maintained as the deck is stepped through and run. DECKBUILD places the text caret on the current line each time the line is reset. You can set the current line anywhere by selecting (highlighting) any part of the desired line and clicking on the **line** button. The **Line** display is then updated to reflect the change.

3.4.6: Pausing, Stopping, and Restarting the Simulator

Click on the **quit** button to send a **quit** command down to the simulator. If there are no buffered commands, the quit is executed immediately, and the simulator exits. DECKBUILD knows when the simulation will end and will print out the following message to the tty subwindow.

```
***END***
```

In this case, you may want to use **kill**, which kills the simulator immediately, or the **stop now** button to stop the execution after the current command. The **stop now** function does not kill the simulator so you can continue by using the **cont** button.

To restart the simulator, click on **restart**. The current simulator is then started. To start a different simulator, choose the new simulator as described under **Main Control**. The **Set Current Simulator** button on the **Control Pad** always starts the current simulator if the **Exec Simulator** option is set.

You can also pause the simulation. Pausing is equivalent to typing a control-Z in the C-Shell: the simulation immediately relinquishes its use of the CPU and stops. Unpausing the simulator causes it to continue from exactly where it left off, and is equivalent to bringing a paused job back to the foreground in C-Shell.

Pausing is frequently used to free up some CPU cycles for some other temporary task. When the task is finished, the simulation can be unpaused. Pause the simulator by clicking on **pause**. The button's title changes to **unpause**. Click on the button once again to unpause the simulator.

3.4.7: Initializing the Simulator

DECKBUILD provides a shortcut to initialize the simulator. Select the name of a structure file to initialize and press **init**. DECKBUILD takes the selected text as a filename, figures out which simulator the file came from, starts that simulator, and executes the proper INITIALIZE statement.

For example, if you were running a ATHENA simulation and saved the structure file ATHENA.str, you can restart ATHENA (if it's not already running) and initialize it with that structure by selecting the text ATHENA.str anywhere on the screen and clicking on **init**. DECKBUILD then automatically starts ATHENA and executes the statement `INIT INFILE=ATHENA.str`. Note that the file may exist in another directory. If so, it's necessary to form the file name properly. If ATHENA.str is in /usr/jdoe/an and running DECKBUILD from /usr, select the file name /usr/jdoe/an/ATHENA.str or jdoe/an/ATHENA.str.

Note: Using the **init** button while selecting a line in the deck, instead of a structure file, will result in initialization from history files, assuming history files are present for the selected line.

3.5: Commands

The primary means of accessing popups that are used to write the input deck is the **Commands** menu. Typically, each item on the menu is associated with a popup window that contains controls used to specify an input deck command. For instance, invoking **Implant...** under ATHENA causes the ATHENA Implant popup to appear.

There are different **Commands** menu for each simulator. The menu always reflects the current simulator shown in the lower right-hand corner of the main frame. Changing the current simulator will cause a different menu to appear as described in the Section 3.2: “DeckBuild Controls”.

3.5.1: Deck Writing Paradigm

In general, DECKBUILD uses one of two ways to write an input deck: either all at once or line-by-line. DECKBUILD uses each as appropriate. Process simulation, for example, is an inherently sequential operation. The same basic commands (`implant`, `diffuse`, `etch`, and `deposit`) are used over and over again. SSUPREM3 and ATHENA are good examples of how this paradigm works, because each popup has a button used to just write the syntax for that popup/command.

3.5.2: Parsing the Deck

DECKBUILD has a built-in feature that allows parsing any part of a deck to automatically configure the appropriate popup(s). For example, to repeat a previous `implant` process command with some minor changes, parse the `implant` statement. To do this, reset the controls of interest on the Implant popup, place the text caret in the proper location, and press the **WRITE** button. To parse any fragment of text, highlight the text and select **Commands**→**Parse Deck...** DECKBUILD scans the highlighted text, determines the proper popups to change, resets the settings of all parameters specified in the highlighted text, and makes the popup(s) visible.

A Few Points to Observe

- **Parse Deck** does not change the settings of parameters that are not specified. If parsing the line `implant boron`, the values of energy and dose, for example, will not be altered from whatever previous value they had on the popup.
- **Parse Deck** parses any highlighted text whether it is in DECKBUILD's own text subwindow or in a separate program. If in DECKBUILD's text subwindow (the usual case), DECKBUILD automatically extends the selection of a partial line to cover a full line.
- Highlight as much text, covering as many command statements, as desired. DECKBUILD configures and brings up all appropriate popups for the current simulator. If you highlight more than one of the same statement, the last has priority.
- DECKBUILD ignores all text that it does not understand.

3.5.3: Process Simulators

Figure 3-12 shows the **Commands** menu for a process simulator (ATHENA).

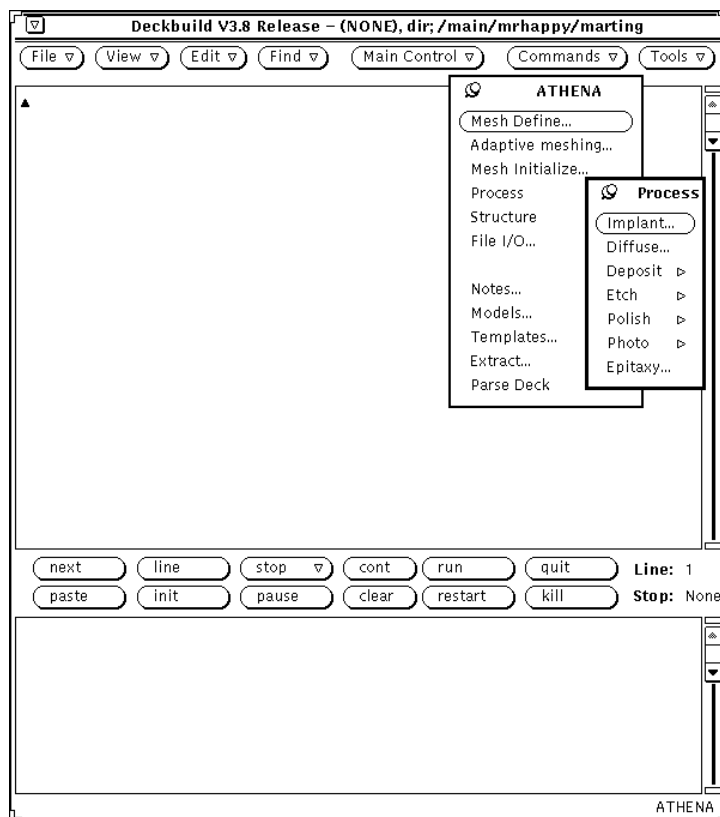


Figure 3-12: Command Menu of ATHENA

Writing a Process Input Deck

Since process fabrication is itself an inherently sequential operation, simply choose the command of interest from the **Commands** menu. A corresponding popup appears that has controls laid out to represent the variable parameters available for the command. For example, Figure 3-13 shows the ATHENA Diffusion popup.

Deckbuild: ATHENA Diffuse

Display: **Time/Temp** Ambient Impurities Models settings

Time/temperature:

Time (minutes): 30 0 500

Temperature (C): 1000 800 1300

End temperature (C): 1300 800 1300

Temperature rate (C/min): 0.000 Rate: Variable

Temp: Constant Ramped

Ambient:

Ambient: Dry O2 Wet O2 Nitrogen Gas Flow

Gas pressure (atm): 1.00 0.00 10.00

HCL %: 0 100

Comment: _____

WRITE Properties ▾

Figure 3-13: ATHENA Diffusion Popup

Selecting the Categories

Some popups, such as Figure 3-13, contain a non-exclusive **Display** setting at the top of the popup in an attempt to conserve valuable screen space. Click **SELECT** in the boxes to display/undisplay the setting of interest. When enabling a setting such as **Impurities**, the entire popup grows vertically to hold the new section. The popup shrinks again when the setting is disabled. Select as many or as few boxes as needed.

Writing the Text

When all of the controls have been adjusted to reflect the process step to be performed, click the **WRITE** button. A line (or sometimes lines) of text is written to the deck at the location of the text caret. If desired, verify the caret's location before clicking **WRITE**, although DECKBUILD automatically detects if the caret is in the middle of a line and moves it if necessary. For more information, see Section 3.3: "Main Control".

Build the entire process deck by invoking the popups as needed from the **Commands** menu, setting the controls, and writing the deck a popup at a time. You can also parse the deck. That is, read a line or lines of syntax from the deck and automatically configure the correct popup(s). For more information, see Section 3.5.2: "Parsing the Deck".

3.5.4: Clever

DECKBUILD also provides a set of command popups for CLEVER and EXACT products. For more information, see the CLEVER or EXACT USER'S MANUALS.

3.6: Tools

DECKBUILD's Tools menu provides the interface to other VWF INTERACTIVE TOOLS: TONYPLOT, MASKVIEWS, and MANAGER (Figure 3-14). In addition, there is a general **Text Editor** for viewing other files such as external simulation decks executed by a **source** statement (see Section 4.12: "SOURCE") from within the current deck.

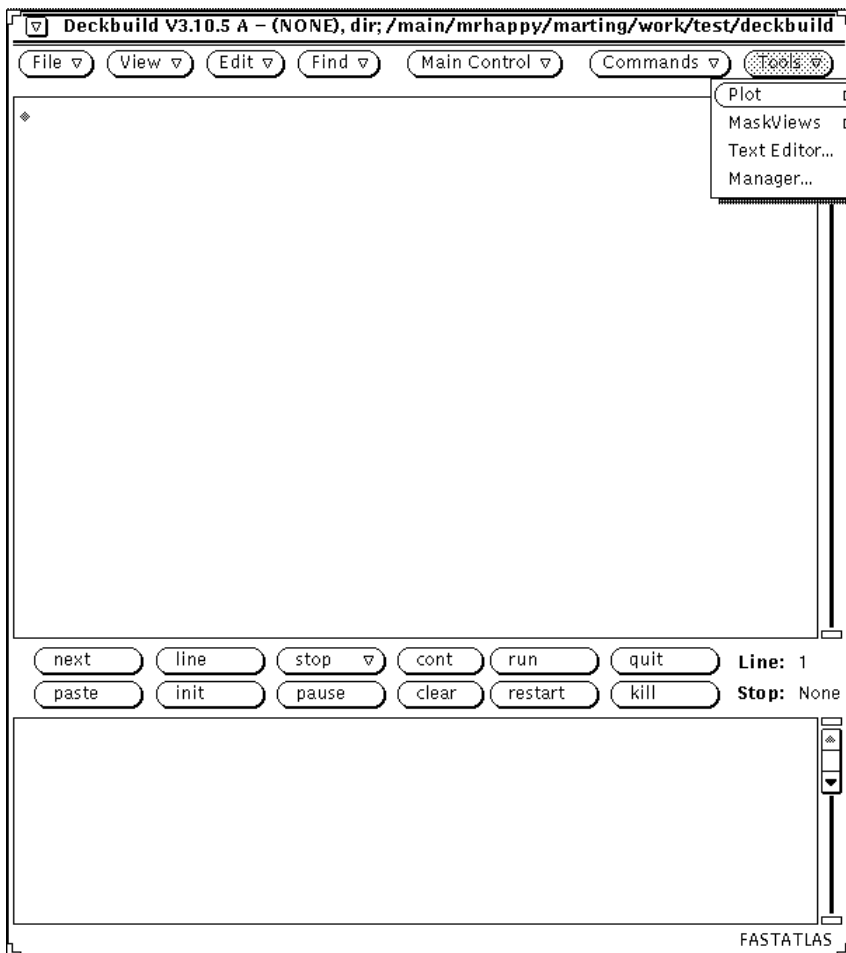


Figure 3-14: The Tools Menu

3.6.1: Starting TonyPlot

Plot simulation results from DECKBUILD by choosing the **Plot structure...** choice in the **Plot** pull-right menu. DECKBUILD allows either plotting the current structure or any specified structure. The following rules are:

- If there is text selected (highlighted) anywhere on the screen, DECKBUILD takes the text as the name of a file to plot. DECKBUILD starts up TONYPLOT on the named file.
- DECKBUILD automatically determines if the selected file is 3D and starts up TONYPLOT3D if appropriate.
- If no text is selected and a simulator is running, DECKBUILD causes the simulator to save its simulation data, then starts TONYPLOT on that data. This does not disrupt any lines from the input deck waiting to be executed by the simulator. For CLEVER, which saves a structure, log file and a layout, two TONYPLOTS and a MASKVIEWS are invoked. Optionally, choose a set file by activating the **Set files...** choice.

Note: The set file is used to record a given plot's layout, such as scaling, zoom, number, and type of plots shown. After a set file is created, it can be used to re-create the same layout when using the same or any other plot data, and often useful for comparing the results of different simulation runs.

The **Plotter Set Files** popup appears (Figure 3-15). This contains a scrolling list of set files in the current directory, and a text field used to search for set files. Adjust the directory name and directory filter if necessary. Click **SELECT** over the name of the desired set file in the scrolling list. If none are desired, ensure that no entries are selected (de-select a selected list entry by clicking **SELECT** on it again). If an entry is selected, it is used as the set file on subsequent plots.

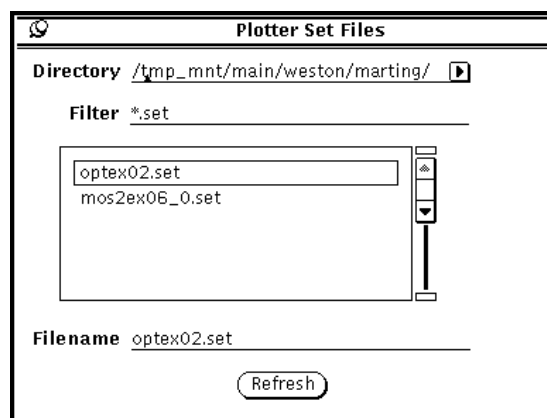


Figure 3-15: Plotter Set Files Popup

Since **Plot structure...** is the default **Tools** menu item, simply click **SELECT** on **Tools** to activate it. This is easier than descending through two levels of menus. Also see **Plot structure** under **Options** on the **Main Control** popup to determine when interactive plots will be made if many lines from the input deck are waiting to be simulated. An option to plot can be set immediately, or at the end of the simulation. The default is immediate.

3.6.2: Starting Maskviews

DECKBUILD allows you to bring up MASKVIEWS with an optional layout file. To choose the layout file, select **Start MaskViews...** from the MASKVIEWS pull-right menu. The MASKVIEWS **Layout Files** popup appears (Figure 3-16). This contains a scrolling list of layout files in the current directory and text fields to search for layout files. Find and select the layout file of choice, then click on the **Start MaskViews** button on the popup (Figure 3-17). After a few moments, MASKVIEWS appears with the specified layout file loaded. If you did not choose a layout file, MASKVIEWS starts with no layout file loaded.

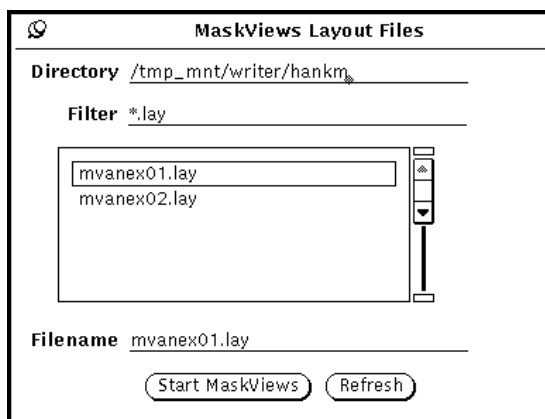


Figure 3-16: MaskViews Layout Files Popup

Loading a Cutline

There are two ways to load a MASKVIEWS cutline from the **MaskViews Cut Files** popup and an alternative method at run time in the simulation deck. To load from the popup, select **Tools→MaskViews→Cut files...** and either save a file from MASKVIEWS and load it into DECKBUILD, or by use the drag-and-drop to drag the cutline directly from the MASKVIEWS previewer. At runtime, you can load a cutline file from the “go simulator” line.

To load a cutline file using the popup:

1. Create and save a cutline file from MASKVIEWS. See the MASKVIEW USER'S MANUAL for more information on how to do this. Typically, the first time through MASKVIEWS would be started from DECKBUILD, create or load a layout, then interactively create a cutline and save it to a file. For later use, you can go straight to step 2.
2. Bring up the **MaskViews Cut Files** popup (Figure 3-17) and set **Category** to **Disk Files**, the default. Choose the cutline file name in the scrolling list. If the file name does not appear, you may need to change the directory and filter on the popup. Click on **Refresh** to refresh the contents of the scrolling list if a new file were just created. You can also enter the name of the file next to **Filename** and click on **Load**.

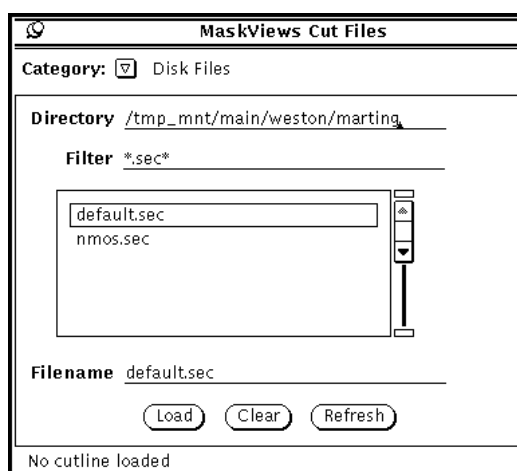


Figure 3-17: MaskViews Cuts Files - Disk Files Category

To load a cutline file via drag-and-drop:

1. Create a cutline file from MASKVIEWS. After either writing or previewing the mask, the cutline masks will be shown on the **2D masks** cutline viewer popup.
2. Bring up the **MaskViews Cut Files** popup (Figure 3-18) in DECKBUILD and set **Category** to **Drag & Drop**. Both this popup and the cutline viewer popup from MASKVIEWS must be visible on your screen.
3. Click and hold the **SELECT** menu button anywhere over the colored masks on the cutline viewer popup in MASKVIEWS. Still holding down **SELECT**, drag the mouse cursor into the large white area in the **Cut Files** popup in DECKBUILD. While dragging, the mouse cursor changes into a special cutline cursor to confirm the process of dragging a cutline. With the cursor over the **Cut Files** popup, release **SELECT**. The cutline information “drops” onto the popup.
4. To load the dropped cutline, click **SELECT** once on the **cutline** icon and click on **Load**. Selected icons are shown surrounded by a square box.

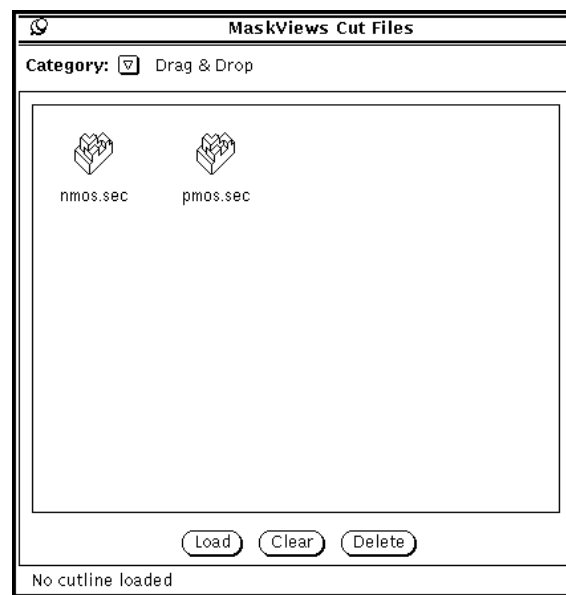


Figure 3-18: MaskViews Cut Files - Drag and Drop Category

To load a cutline from the input deck:

1. Create and save a MASKVIEWS file as specified in step 1 for loading a cutline disk file using the popup.
2. Use the syntax “cutline=filename” in the “go simulator” line to load the previously saved file. The following line loads a cutline stored in the file /default.sec and starts ATHENA.

```
go athena cutline="/usr/jdoe/default.sec"
```

It is possible to drag and drop up to 16 different cutlines this way (that’s all there is room for in the icon drop area). You do not need to save the cutlines to a file to use drag-and-drop. But if the same cutline is to be used again in the future, then it needs to be saved. Save cutlines from MASKVIEWS by clicking on **Write** on its popup (Figure 3-19). Either method of loading a cutline loads the mask information into DECKBUILD, and causes the mask names to appear in the SSUPREM3 and ATHENA **Mask** popups (Figure 3-22).

Note: You can clear the currently loaded cutline by either selecting the clear button in the MaskViews Cut Files popup or by using the “cutline=none” syntax in the “go simulator” line.

3.6.3: Starting Text Editor

Use the **Text Editor...** choice to startup the general **Text Editor** application with the file currently highlighted within the deck. The Editor is only invoked if a valid filename is provided. Otherwise, an error is displayed.

3.6.4: Starting Manager

Use the **Manager...** choice to start up the VWF INTERACTIVE TOOLS MANAGER. The menu item is placed here as a convenience feature.

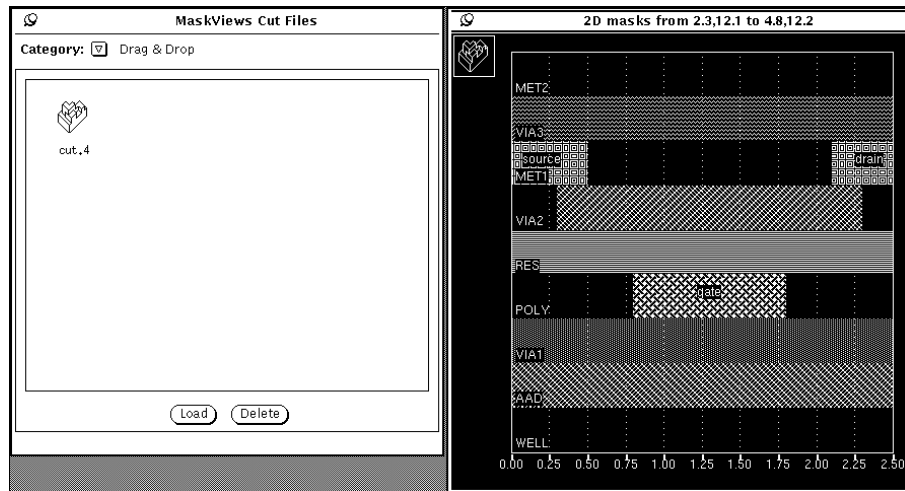


Figure 3-19: Drag and Drop from MaskViews

3.7: History

3.7.1: Overview

The **History** function allows moving backwards to any previous line in the input deck and restart execution. It is especially useful when debugging new decks, performing “what if” simulations, and in visualizing the device at different stages in the process flow.

DECKBUILD maintains a set of history files saved from the simulator as the simulation progresses. This permits going back to any previous step in the process by simply clicking on a line in the input deck. DECKBUILD automatically re-initializes the simulator with the correct history file.

3.7.2: History Control

You can configure how and if DECKBUILD maintains history files with the **History** popup. Click the **History Props...** button from the **Control Pad** category on the **Main Control** popup and the History popup appears (see Figure 3-20).

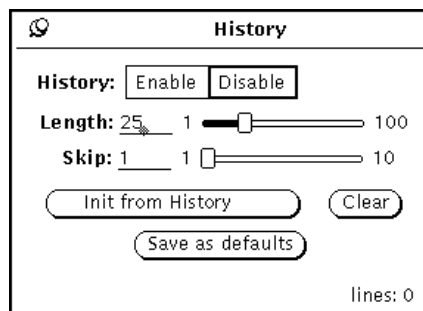


Figure 3-20: The History Popup

The **History** setting turns the entire mechanism on or off. If enabled, DECKBUILD saves history files after each significant simulation step. If disabled, history files are not saved and the simulation runs a little faster. History can be turned on and off as the simulation progresses. DECKBUILD allows re-initializing from any part of the deck that was run while history was enabled. History is enabled by default.

Length determines how many trailing process flow steps to remember (how many history files to maintain). The default is 25, although up to 100 steps may be saved. When the limit is reached, DECKBUILD starts re-using the old history files in a loop.

Skip determines how often history files are saved. The default value of 1 indicates that every significant process step causes a history file to be saved. A value of 2 indicates every other step, and so on. Significant process steps are implant, diffuse, etch, deposit, initialize/load, profile, and certain other statements. Comments, plot statements, blank lines, and certain other details are ignored. History files are not saved during device simulation.

Path, if activated, specifies the directory that the saved history files are saved in. This may be useful if large structures are being simulated and disk space for the current working directory is not abundant.

Note: If a simulation is executed with history and the path then altered, history initialization for the previous simulation fails.

Compress is switched off, by default, but when on all history files are compressed and appears in the form `history%.str.gz`. These files are automatically decompressed for initialization from history or loading into TONYPLOT. This function may be useful when simulating large or complex structures.

Clicking **Save as defaults** saves the current settings for use the next time DECKBUILD is run.

Initializing From History

After running part, or all the way through the deck with history enabled, the simulator can be re-initialized in the state it was in at some previous point in the deck. Re-initialize by selecting (highlighting) the line of interest, then clicking on the **Init** button on the Execution Control panel on the main window between the text and tty subwindows. For backwards compatibility, the **Init from History** button on the Main Control popup also provides this feature.

Note: This overloads the functionality of the Init button: if you select a filename which exists in the current directory instead, DECKBUILD causes the simulator to load that file).

DECKBUILD also resets the current line to the selected line.

DECKBUILD may not have any history attached to the selected line if a comment line has been selected or skipped. DECKBUILD displays a notice prompt and suggest a previous point in the input deck by highlighting it. If selecting a line that is so far back in the deck that DECKBUILD no longer maintains relevant history, a notice prompt appears to inform you of the condition. Use a line closer to the current position or increase the history length.

When re-initializing from history, any “go simulator” flags (see the Sections 3.8: “Auto Interfacing” and 4.6: “GO”) specified on the go statement associated with the selected line are also re-initialized. For example, if a MASKVIEWS cutline file had previously been loaded using the syntax cutline=filename, then the specified file would be reloaded into DECKBUILD.

Removing History Files

Since history files can take up a fair amount of storage space, DECKBUILD provides two ways to remove them. First is to delete them at any time during the run by clicking on **Clear** in the **History** popup. Second, let them be removed when DECKBUILD is quit.

DECKBUILD is configured by default to remove history files at quit time, and displays a notice prompt to confirm their deletion. Change the default by changing the **Remove history files** setting on the **Options** category of the **Main Control** popup. The choices are to remove history files, confirm their deletion, or not to remove them at all.

In any case, history files are always saved in and removed from the current directory.

3.8: Auto Interfacing

Auto interfacing is the term used to describe DECKBUILD's capability of automatically transferring simulation data between different simulators. Simulation of a device may proceed transparently from SSUPREM3 (1D process), through ATHENA (2D process), and finally to ATLAS (2D device). The thread of control can be transferred to any simulator under DECKBUILD, including DEVEDIT for interactive mesh adaptation. Auto Interfacing enhances the power of simulation by allowing concentration on which simulator is best for the job, rather than on how to get one simulator to talk to another. SILVACO standards of simulator commonality is based on products that use a common data format.

3.8.1: Scenario

A typical simulation flow is shown in Figure 3-21.

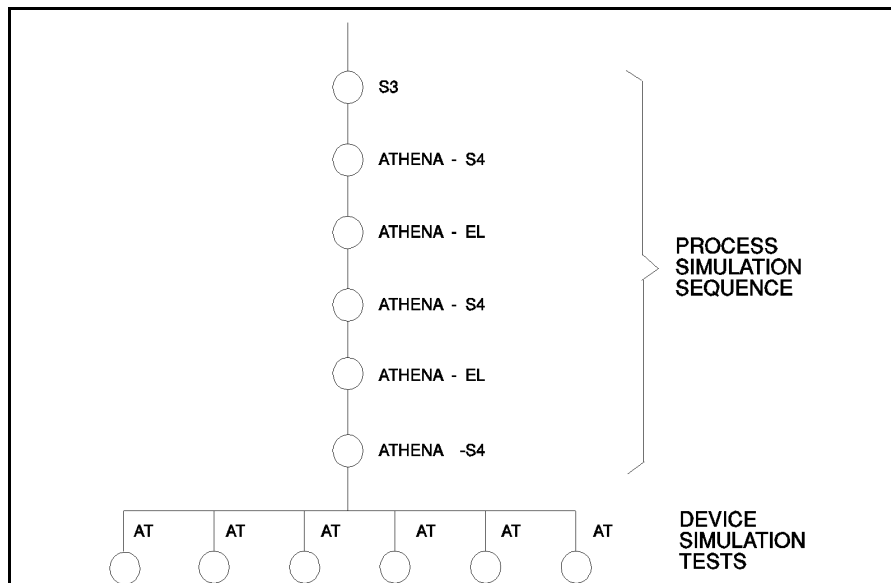


Figure 3-21: The History Popup

The 2D simulation starts with a 1D process simulator (SSUPREM3), since the initial processing of the device is entirely planar. For example, the initial well processing of a MOS device can be considered in 1D until the gate poly is etched. At the point when 2D is first required, an auto interface statement is placed in the deck, followed by 2D mesh definition and mesh initialization commands. At run time, DECKBUILD automatically transfers the 1D data from SSUPREM3 onto the 2D ATHENA mesh.

Alternately, the auto mode of ATHENA can be used. This mode automatically performs a 1D simulation in ATHENA until any statement requiring 2D is encountered (such as an etch left/right). ATHENA then transfers automatically into 2D mode for the remainder of the simulation. You benefit from applying the same models and syntax throughout the process simulation. The choice of SSUPREM3 or 1D ATHENA depends on which simulator is preferred, or for which special models have developed.

Once in ATHENA, you can continue process simulation and interface to other 2D simulators as well. It is also possible to interface to DEVEDIT at any time to adaptively remesh the device in preparation for device simulation.

Finally, a number of ATLAS device tests are shown at the end of the process sequence. Unlike the process sequence, where each section acts as a link in the chain of processing, the device tests each act on the final process structure. Append as many device tests as needed to the end of a process simulation and each will use the same final process structure as input.

The Active Structure

This final structure at the end of process simulation is called the active structure. DECKBUILD saves and remembers the active structure whenever auto interfacing is performed from a process simulator to any other simulator. Device tests always use the active structure unless explicitly initialized otherwise.

The current active structure is always shown on the left footer of the Main Control popup.

The Auto Interface Statement

The place in the input deck where auto interfacing should occur is marked by inserting an auto interface statement. The statement looks like

```
go simulator
```

where simulator is any valid simulator name. Consider the following input deck fragment that interfaces from SSUPREM3 to ATHENA:

```
GO SSUPREM3
#
INIT SILICON THICK=1.2 SPACE=500 BORON CONC=1E14
#
DIFFUSE TEMP=100 TIME=20 WETO2
#
IMPLANT PHOSPHORUS DOSE=1E13 ENERGY=40 PEARSON
#
GO ATHENA
#
LINE Y LOC=0.0 SPAC=0.2 TAG=TOP
LINE Y LOC=0.50 SPAC=0.10
LINE Y LOC=1.00 SPAC=0.15 TAG=BOTTOM
#
LINE X LOC=0.00 SPAC=0.10 TAG=LEFT
LINE X LOC=1.00 SPAC=0.10 TAG=RIGHT
#
INIT ORIENTATION=100 AUTO
#
ETCH OXIDE RIGHT P1.X=0.2
```

Here, an oxide in 1D is grown using SSUPREM3 and transfers control to ATHENA to perform the 2D etch. At run time, the 1D doping profile is automatically transferred from SSUPREM3 onto the 2D ATHENA mesh, and oxide deposited on top. The oxide profile is transferred.

Probably the best way to create an auto interface statement is to have DECKBUILD create it automatically. This is done by placing the text caret in the text subwindow at the point desired to insert the statement. Then, enable the **Write to Deck** choice on the **Control Pad** and click on **Select Current Simulator** to write the auto interface statement. Usually, the most convenient time to do this is when finished writing statements for one simulator and beginning to write statements for the

next. DECKBUILD does not only insert the auto interface statements but also brings up the proper **Commands** menu for the new simulator.

How Auto Interface Works

When DECKBUILD gets a request to perform an auto interface (from an `auto interface` statement in the input deck or through the **Control Pad**), it evaluates whether an interface is appropriate: 1D to 2D process is legal, but 2D to 1D process is not. If an interface is appropriate, then DECKBUILD also checks to see if the current simulator has been initialized or not. For example, if ATHENA has not yet executed an `initialize` statement, it doesn't have any simulation results to pass on to the next simulator. Finally, if both these conditions are satisfied, then DECKBUILD causes the current simulator to save its simulation data, shut down the current simulator, starts up the new simulator, and initializes the new simulator with the saved data. If either condition is not satisfied, then DECKBUILD honors the request to start up the new simulator, but does not attempt to initialize it with saved simulation data. The latter is appropriate when moving backwards in an input deck, and for quick "look and see" experiments with another simulator.

You can also alter the default input and output flags using the `go simulator interface` statement. For example, the default `load` statement for DEVEDIT includes the **mesh** flag. Using the following syntax, the mesh can be loaded on auto interface:

```
go devedit inflags = "mesh"
```

Simulator flags can be appended to the existing default flags and MASKVIEWS cutlines can also be loaded automatically in the deck using the **simflags** and **cutline** arguments respectively. For more information and examples, see Section 4.6: "GO".

3.9: IC Layout Interface

The **IC Layout Interface** (MASKVIEWS) allows the building of a deck that can be used to run a cross-section from any region on the layout. Such a deck is called a generic deck. A generic deck is always used in conjunction with the **IC Layout Interface**, which consists of loading cutline information into DECKBUILD. The cutline information contains location-specific masking information from a 1D or 2D cutline across the surface of the layout, taken from MASKVIEWS.

To use the **IC Layout Interface**, create a layout and a corresponding generic deck. The generic deck uses mask information defined on the layout and is identical in nature to the “run sheet” used in the fab. Unlike the typical process simulation input deck, it defines the order of process steps and interweaves the mask steps.

When the generic deck is complete, choose a one or two-dimensional cross-section in MASKVIEWS over the layout. This cross-section is known as a cutline. Load the cutline into DECKBUILD and click on the **Run** button. DECKBUILD automatically uses the cutline information to substitute mask, mesh, region, and electrode information at run time.

For information on how to start MASKVIEWS and load cutlines, see Section 3.6: “Tools”. For information on how to create a layout, see the MASKVIEWS USER’S MANUAL.

3.9.1: Creating a Generic Deck

Generic decks have no geometry information, use masks, optionally use regions and electrodes, and have no horizontal grid information for 2D mesh generation. All of this information is quite specific to the cross-section that has been chosen through the layout and is contained in the cutline file. The horizontal grid information is calculated by MASKVIEWS so that a line with a user-specified grid spacing is placed at each mask edge and is substituted by DECKBUILD at run time.

Mask Statements

The best place to start writing a generic deck is with mask definitions. The first step is to initialize the Mask popup with the correct mask names to write the deck properly. To do this, invoke MASKVIEWS from DECKBUILD using the **Tools** menu, then create or load the layout. Create a cross-section and store it to a cutline file, then load that file into DECKBUILD as shown in Section 3.6: “Tools”.

At this point, the **Mask** popup (Figure 3-22) should display all the masks in a scrolling list. Pinning the popup keeps it from disappearing each time a mask is written to the deck.

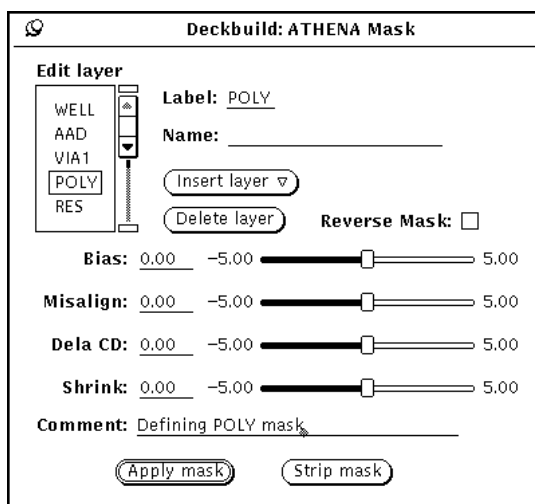


Figure 3-22: The Mask Popup

Write the deck as much as it the process flow appears in the fab. Use masks to structure the deck so that it is capable of producing any device on the layout. Insert mask statements by selecting the desired mask on the scrolling list, then clicking on **WRITE**.

At runtime, the mask statements are substituted with either deposit or etch or both statements or **Optolith** layout statements when the optolith flag is used with the mask command

MASKVIEWS allows the definition of either barrier or photoresist as the masking material. Barrier is a fictitious material that behaves like a perfect photoresist and can be deposited in very thin layers (about 0.02 microns) to save grid points. Barrier is recommended in all situations except when photoresist penetration studies are required.

3.9.2: Regions

You can also define regions in MASKVIEWS. Regions are boolean combinations of masks that uniquely define an area on the layout. Regions are used in extract statements in place of X.VAL to define the x-location where the desired quantity is to be extracted. For example, a layout for a MOS inverter that contains n- and p-type devices may have WELL, AAD, and POLY masks. To measure the gate oxide thickness in both types of devices, one approach might be to define a region GATE in MASKVIEWS where WELL is "don't care", AAD is "true", and POLY is "true". All other masks would be "don't care". Then, use this region in an extract statement:

```
extract oxide thickness region="GATE"
```

rather than:

```
extract oxide thickness x.val=1.0
```

Electrodes

Electrode positioning is the last remaining area of the deck that requires layout-specific values. In non-generic decks, an electrode statement in ATHENA requires both a specific x-location value and a name for that electrode. For example:

```
electrode name="gate" x.val=1.0
```

A generic deck can obviously assume neither the x-location of an electrode, its name, nor even how many electrodes there are. The autoelectrode statement provides the solution to automatically place and name electrodes.

Use MASKVIEWS to define which masks are electrode masks and the corresponding electrode names. Electrode masks are those masks that are used for defining one or more electrodes. For instance, the POLY mask for a MOS device is an electrode mask because it forms the gate contact. Along with the electrode attribute, MASKVIEWS allows the specifying of an electrode name for a mask (or part of a mask) as well. Electrode names and masks should be specified in MASKVIEWS before generating a cutline file in MASKVIEWS.

To use the electrode masks, enter the autoelectrode statement in the input deck directly after contact definition using the mask of interest. Autoelectrode takes no parameters or arguments. It works by inserting electrode statement(s) at run time using information corresponding to the last electrode mask. If the same mask is used to define more than one contact, use MASKVIEWS to assign a separate name for each section of the mask. DECKBUILD substitutes a separate electrode statement for each contact. Thus, a single autoelectrode can generate multiple electrode statements.

Figure 3-23 shows the use of autoelectrode in a generic deck. Notice that DECKBUILD automatically comments out the mask and autoelectrode statements from the deck as they are executed by the simulator. The comments only appear in the run-time output; the deck itself is not changed.

Note: DECKBUILD only remembers the electrodes specified within each mask. Therefore, an autoelectrode statement must be used for every mask layer where electrodes are defined. This defines multiple electrodes for a single autoelectrode statement within the current mask. In other words, both the source and drain of a MOS transistor could be located on the same metal level.



Figure 3-23: DeckBuild Main Window

If structure files are saved after masking and electrode steps, they must be saved after the autoelectrode statement, and not in between the mask statement and the autoelectrode statement. Otherwise, the structure file does not contain the electrode information (which is inserted by the autoelectrode statement). In this case, the only way to add it would be to go through the masking operation a second time as shown in the following example.

```
mask name="POLY"

etch poly dry thick=0.4

strip

autoelectrode

struct outf="poly.str"
```

Do not do this:

```
mask name="POLY"

etch poly dry thick=0.4

strip

struct outf="poly.str"

autoelectrode
```

Enabling

Enable cutline substitution by setting **Auto Maskviews** to **ON** on the **Main Control Options** popup. Substitution begins as soon as a cutline file is loaded from the **MaskViews Cut Files** popup. This popup can be accessed from the **Tools** menu. See Section 3.6: "Tools" for information on loading a cutline.

Disable substitution by turning **Auto MaskViews** to **OFF**. No substitution occurs even if a cutline file is loaded.

Cutlines can also be loaded and cleared from DECKBUILD at runtime using the go simulator cutline=filename syntax. DECKBUILD loads the MASKVIEWS cutline from the current working directory if you do not specify path. DECKBUILD clears an existing cutline if you specify cutline=none.

3.9.3: Rules of Thumb

To make sure that your deck is indeed generic, obey the following rules:

1. etch statements are either

```
etch dry material thickness value
```

or

```
etch material all
```

2. Do not use

```
etch left material pl.x=value
```

3. Do not include horizontal mesh information for 2D process simulators. The mesh information is substituted automatically at run time.

4. Use mask statements where photoresist is required:

```
mask name="PWELL"
```

or

```
mask name="AAD" reverse
```

5. Beware of wedges. As in real experiments, wedges, fillets, and spacers may appear if anisotropic (dry) etches are not used carefully.

6. Use region in extract statements rather than x values.

```
extract oxide thickness region="GATE"
```

7. Use autoelectrode statements rather than electrode statement

3.9.4: Mask Bias, Misalignment, and Delta CD

Mask, bias, misalignment, and delta CD information can be entered in the mask statement to study these effects, either standalone or as experimental variables in the VWF environment.

Bias, misalignment, and delta CD can be studied together or independently. Misalignment shifts the entire specified mask left (negative numbers) or right. Bias and delta cd increase or decrease the width of the mask (for positive masks, a positive bias/delta cd will decrease the width of the etched hole(s) in the mask).

A value of 0 for any parameter is equivalent to not specifying that parameter. The values have units of microns.

The syntax is:

```
mask name="<region_name>" [bias=<value>]/  
[misalign=<value>] [delta_cd=<value>]
```

3.9.5: Using DevEdit with IC Layout

DEVEDIT can use mask region information when running under DECKBUILD to place mesh constraints in areas specified by MASKVIEWS regions. An example would be to place a finer mesh in the region under the gate of a MOS device. This can be specified from the **Mesh Constraints** popup of DECKBUILD. The resulting syntax is:

```
constrain.mesh under.mask="regionname" other_parameters...
```

Note: `under.mask` is a misnomer. The specified area is a MASKVIEWS region, which is a boolean combination of perhaps several masks, so the region may not be wholly under or composed of a single mask. It is really under a region, not a mask. To eliminate confusion with DEVEDIT material regions, use the parameter `under.mask`.

3.10: UTMOST Interface

The UTMOST interface allows SILVACO's parameter extraction package UTMOST III to load in data from one of more device simulation runs and perform SPICE model parameter extraction. With the VWF, in particular, this allows you to generate response surfaces that model SPICE parameters as a function of process variations, including study of failure analysis, process synthesis, yield analysis, and with the SPAYN interface, worst case modeling.

3.10.1: Setting Up An UTMOST Input Deck

Although UTMOST normally runs as an interactive X-based program, you can run without its graphical interface and read commands from an interactive prompt when run under DECKBUILD like the other simulation tools.

UTMOST Input Deck Example

```
##### START UTMOST SIMULATION #####
go utmost
utmost type = mos
#
load Utmost model filemodel bsim3_pmos
#
set value for TOX
set tox_in_m = $tox*1.0e-10
device TOX = $tox_in_m
#
define device specifications
setup width = 1.0 length = 0.6 polarity = P
#
load Atlas log files
init inf= IdVg-Vb.log master
init inf= IdVd-Vg.log master append
#
select required characteristics for device
deselect ID/VG-VB all
select ID/VG-VB device 1
deselect ID/VD-VG all
select ID/VD-VG device 1
log outf = mos.log utmost
#
perform simulation
fit ID/VG-VB
fit ID/VD-VG
```

```
##### EXTRACT UTMOST PARAMETERS #####
```

```
extract name="P-VTH0" param="VTH0"  
extract name="P-K1" param="K1"  
extract name="P-K2" param="K2"  
extract name="P-K3" param="K3"  
extract name="P-W0" param="W0"  
extract name="P-NLX" param="NLX"  
extract name="P-DVT0" param="DVT0"  
extract name="P-DVT1" param="DVT1"  
extract name="P-UA" param="UA"  
extract name="P-UB" param="UB"  
extract name="P-UC" param="UC"  
extract name="P-VSAT" param="VSAT"  
extract name="P-A0" param="A0"  
extract name="P-A1" param="A1"  
extract name="P-A2" param="A2"  
extract name="P-RDSW" param="RDSW"  
extract name="P-VOFF" param="VOFF"  
extract name="P-NFACTOR" param="NFACTOR"  
extract name="P-PCLM" param="PCLM"  
extract name="P-PDIBL1" param="PDIBL1"  
extract name="P-PDIBL2" param="PDIBL2"  
extract name="P-DROUT" param="DROUT"  
extract name="P-PSCBE1" param="PSCBE1"  
extract name="P-PSCBE2" param="PSCBE2"  
extract name="P-TOX" param="TOX"  
extract name="P-XJ" param="XJ"  
extract name="P-U0" param="U0"  
extract name="P-ETA" param="ETA"  
extract name="P-Ilinm" param="Ilinm"  
extract name="P-Ilins" param="Ilins"  
extract name="P-Isatm" param="Isatm"  
extract name="P-Isats" param="Isats"
```

```
quit
```

There are several points that must be followed to set up an input deck.

The first non-comment statement after the `go utmost` command must be `UTMOST TYPE=type` where `type` is either MOS, BIP, DIODE, GAAS, or JFET.

When the `UTMOST` statement is encountered, the correct `UTMOST` module is executed. If no `UTMOST` statement is encountered, the MOS module is run by default.

Note: The `UTMOST` command (`utmost type=type`) can be replaced by specifying the module type command line flag in the `go utmost` as below. This will append the `"-bip"` flag to the default `UTMOST` argument and start the correct module immediately. For example, `go utmost simflags="-bip"`.

The next non-comment statement must be a `model` statement. The `model` statement gives the model file name that `UTMOST` reads. The model file is assumed to exist in the `$SILVACO/var/utmost` directory. Therefore, the statement

```
MODEL BSIM3_P MOS
```

reads in the file `$SILVACO/var/utmost/bsim3_p.mos`.

To create this model file, run a baseline device simulation, load its results (IV curves) into `UTMOST` interactively, perform the necessary modeling to generate the desired parameters, then save the model file from `UTMOST`. Once this has been done with a baseline device, the same model file can be used for similar devices. For example, a large process variation experiment on a device in the VWF can use the same model file for testing all variations.

Note: You need different model files for different devices. In other words, *n* versus *p* MOSFETs.

Use the `INIT` command to explicitly load one or more simulated IV result files:

```
INIT INF=filename MASTER append
```

The master flag tells `UTMOST` that the data is in master, or SSF, file format (the default output format from ATLAS). Use the `append` flag for loading all but the first IV file, as shown above.

Note: Load the files explicitly. The **auto-interface** feature of `DECKBUILD` is not active in `UTMOST` mode, and does not load them automatically.

Place `extract` statements at the end of the deck to extract the modeled SPICE parameters. The format of the `extract` statement is:

```
extract name="name" param="utmost_parameter_name"
```

where `name` is any name of your choice, and `utmost_parameter_name` is the name of an `UTMOST` parameter.

It is helpful to first run the deck without any `extract` statements, but with a `save outfile=filename` command. `UTMOST` always prints a list of all parameters it has modeled before it saves the file. You can then copy and paste directly from this list into your `extract` statements. The parameter names are case sensitive and the `extract` parameter name should match the `UTMOST` parameter name exactly.

Another noteworthy point in the example concerns the device TOX setting, which is taken as the value of `$tox` multiplied by a conversion factor. `$tox` is an extracted value taken from a gate oxide thickness measured in the ATHENA process simulation in the same input deck. `$tox` is also used in a `set` statement to convert its units from angstroms (always used by `extract` to measure material thicknesses) to meters (used by `UTMOST`). The `set` statement performs the arithmetic, not in a simulator statement. Finally, the value of the result is substituted in the device TOX command.

In the case of the VWF, the `$tox` thickness is measured in an input deck fragment run separately from the `UTMOST` input deck. The VWF remembers the value of `$tox` (and all other extracted variables) and passes it down to all other children fragments. Therefore, it can still be used at any later point in the simulation thread.

In the VWF environment, the `init` infile filenames are substituted with automatically-generated filenames that contain all IV simulation data from all device simulations that feed into the `UTMOST` test. See the VWF manual for more information on how to connect device tests to `UTMOST` tests. It is important to note that the VWF substitutes all I-V data sets saved from all connected device tests (all data that was saved with a `'log outfile'` command in the device tests). Unusual results may occur in the case where many extraneous files have been saved in the device tests. In that case, remove the extraneous `log outfile` statements as necessary.

Runtime Output Example

```
UTMOST> ##### START UTMOST SIMULATION #####
UTMOST> utmost type = mos
U T M O S T III
P A R A M E T E R E X T R A C T I O N S O F T W A R E
Version: 10.04 (Batch-mode)
Preliminary Version
Copyright 1989, 1990, 1991, 1992, 1993, 1994
SILVACO Data Systems
All rights reserved
=====
MOS Module: enabled
BIP Module: disabled
JFET Module: disabled
Diode Module: disabled
GAAS Module: disabled
Fitting Routines : enabled
Local Optimization : enabled
Global Optimization : enabled
Simulation : enabled
=====
Fri Jul 29 16:25:36 1994
Executing on host: elvis
UTMOST>
```

```
UTMOST># load UTMOST model file
UTMOST>model bsim3_pmos
  SETUP FILE LOADED. Version number: 66
UTMOST>
UTMOST># set value for TOX
UTMOST>## set tox_in_m = $tox*1.0e-10
UTMOST>device TOX = 1.79e-08
UTMOST>
UTMOST># define device specifications
UTMOST>setup width = 1.0 length = 0.6 polarity = P
UTMOST>
MOST># load Atlas log files
UTMOST>init inf= IdVg-Vb.log master
UTMOST>init inf= IdVd-Vg.log master append
UTMOST>
Log file loaded
UTMOST># select required characteristics for device
UTMOST>deselect ID/VG-VB all
UTMOST>select ID/VG-VB device 1
UTMOST>deselect ID/VD-VG all
UTMOST>select ID/VD-VG device 1
UTMOST>UTMOST>log outf = mos.log utmost
UTMOST>
UTMOST># perform simulation
UTMOST>fit ID/VG-VB
  Please wait, FITTING in progress!
  Executing local optimization ivgs_bsim3_a
  Executing local optimization ivgs_bsim3_a
UTMOST>fit ID/VD-VG
  Please wait, FITTING in progress
  Executing local optimization ivds_bsim3_a
  Executing local optimization ivds_bsim3_a
UTMOST>
UTMOST># output UTMOST parameters for extraction (used only for setup)
UTMOST>#save outf = mos.ssf
UTMOST>
UTMOST>##### Extract UTMOST parameters #####
```

```

UTMOST>
UTMOST>save outfile=/tmp/deckbEAAa06379
The following parameters have been stored:
VTH0   = -0.6363853      [V]           K1      = 0.9019918      [V^0.5]
K2      = -0.0688711     [.]           K3      = 65.72977       [.]
W0      = 4.883524E-6    [m]           NLX     = 4.674296E-8    [m]
DVT0    = 3              [.]           DVT1    = 0.452118     [.]
UA      = 6.048951E-10   [m/V]        UB      = 1E-22         [(m/V)^2]
UC      = -0.0253118     [1/V]        VSAT    = 5.8867E6      [cm/sec]
A0      = 0.5285819      [.]           A1      = 0.0539283     [1/V]
A2      = 0.6716171      [.]           RDSW    = 800           [ohm*m^3]
OFF     = -0.0139744     [V]           NFACTOR = 1.3974359     [.]
PCLM    = 5.7002197      [.]           PDIBL1  = 0.0545063     [.]
PDIBL2  = 0.0610933      [.]           DROUT   = 0.1460536     [.]
PSCBE1  = 9.97005E9      [V/m]        PSCBE2  = 1E-9           [V/m]
TOX     = 1.79E-8        [m]           XJ      = 1.5E-7        [m]
U0      = 213.4599111    [cm^2/V/sec]  ETA     = 0.3            [.]
Ilinm   = 2.009946E-5    [A]           Ilin    = 1.99748E-5     [A]
Isatm   = 2.773636E-4    [A]           Isats   = 2.76221E-4     [A]

UTMOST>
EXTRACT> init infile="/tmp/deckbEAAa06379"
EXTRACT> extract name="P-VTH0" param="VTH0"
P-VTH0=-0.636385 V
EXTRACT> extract name="P-K1" param="K1"
P-K1=0.901992 V^0.5
EXTRACT> extract name="P-K2" param="K2"
P-K2=-0.0688711
EXTRACT> extract name="P-K3" param="K3"
P-K3=65.7298
EXTRACT> extract name="P-W0" param="W0"
P-W0=4.88352e-06 m
EXTRACT> extract name="P-NLX" param="NLX"
P-NLX=4.6743e-08 m
EXTRACT> extract name="P-DVT0" param="DVT0"
P-DVT0=3
EXTRACT> extract name="P-DVT1" param="DVT1"
P-DVT1=0.452118
EXTRACT> extract name="P-UA" param="UA"

```

```
P-UA=6.04895e-10 m/V
EXTRACT> extract name="P-UB" param="UB"
P-UB=1e-22 (m/V)^2
EXTRACT> extract name="P-UC" param="UC"
P-UC=-0.0253118 1/V
EXTRACT> extract name="P-VSAT" param="VSAT"
P-VSAT=5.8867e+06 cm/sec
EXTRACT> extract name="P-A0" param="A0"
P-A0=0.528582
EXTRACT> extract name="P-A1" param="A1"
P-A1=0.0539283 1/V
EXTRACT> extract name="P-A2" param="A2"
P-A2=0.671617
EXTRACT> extract name="P-RDSW" param="RDSW"
P-RDSW=800 ohm*m^3
EXTRACT> extract name="P-VOFF" param="VOFF"
P-VOFF=-0.0139744 V
EXTRACT> extract name="P-NFACTOR" param="NFACTOR"
P-NFACTOR=1.39744
EXTRACT> extract name="P-PCLM" param="PCLM"
P-PCLM=5.70022
EXTRACT> extract name="P-PDIBL1" param="PDIBL1"
P-PDIBL1=0.0545063
EXTRACT> extract name="P-PDIBL2" param="PDIBL2"
P-PDIBL2=0.0610933
EXTRACT> extract name="P-DROUT" param="DROUT"
P-DROUT=0.146054
EXTRACT> extract name="P-PSCBE1" param="PSCBE1"
P-PSCBE1=9.97005e+09 V/m
EXTRACT> extract name="P-PSCBE2" param="PSCBE2"
P-PSCBE2=1e-09 V/m
EXTRACT> extract name="P-TOX" param="TOX"
P-TOX=1.79e-08 m
EXTRACT> extract name="P-XJ" param="XJ"
P-XJ=1.5e-07 m
EXTRACT> extract name="P-U0" param="U0"
P-U0=213.46 cm^2/V/sec
```

```
EXTRACT> extract name="P-ETA" param="ETA"
P-ETA=0.3
EXTRACT> extract name="P-Ilinm" param="Ilinm"
P-Ilinm=2.00995e-05 A
EXTRACT> extract name="P-Ilins" param="Ilins"
P-Ilins=1.99748e-05 A
EXTRACT> extract name="P-Isatm" param="Isatm"
P-Isatm=0.000277364 A
EXTRACT> extract name="P-Isats" param="Isats"
P-Isats=0.000276221 A
EXTRACT> quit
UTMOST>quit
UTMOST finished
*** END ***
```

3.11: SmartSpice Interface

The SMARTSPICE interface allows SILVACO's circuit simulation program to execute within DECKBUILD and the VWF AUTOMATION TOOLS. The interface reads through the whole deck \$-substituting any variables that have been set. The actual simulation does not occur until the solve outfile=<rawfile> is reached in the simulation deck.

The output rawfile is a Data Format file that can be visualized in TONYPLOT or used with extract statements to obtain required measurements as shown below.

```
extract init infile="spice.dat"

extract name="curve1"

max (curve(da.value."vin", da.value."power"))

extract name="curve2"

max (curve(da.value."2" "vin", da.value."2" "power"))
```

These extract statements will return the maximum of power for the first and second data set in the file spice.dat.

3.12: Internal Interface

The **Internal** interface is provided as a split point area for **Device** simulation only experiments using VWF AUTOMATION TOOLS. This interface only accepts certain DECKBUILD statements (set, tonyplot, extract, go, source), the most useful is the set command. The **Internal** interface is intended to include set statements which define the required input values to be \$-substituted into the device experiment. Using VWF AUTOMATION TOOLS, these inputs can be split on (varied) over many simulations to provide Device experiments without using process simulation.

3.13: Remote Simulation

DECKBUILD has the capability to be running on a local host while executing a simulation on a remote host. The simulation is run using a remote shell command while displaying the output back to the tty window in DECKBUILD.

In interactive mode, the remote host for each simulator can be specified using the Simulator Properties popup accessed from the Main Control popup. For batch mode (-run), you can use the -remote <hostname> command line option. This specifies the same remote host for all simulators used.

3.13.1: Remote Options

Within the **Main Control** popup under the Options category, there are some remote options that can be used to customize remote simulation.

Remote tmp directory sets the remote simulation tmp directory, which must be mounted on the host executing DECKBUILD and the host executing the simulator.

Remote shell command specifies the remote shell command to be used for remote simulation. This option may need to be set with a specific path such as:

```
Solaris2 & decalpa-osf1 /bin/rsh
Solaris1 /usr/ucb/rsh
rs6000-aix4 /usr/bin/rsh
hp700-hpux /usr/bin/remsh
mips64-irix6 /usr/bsd/rsh
```

Remote mount string removes the automount prefix (usually /tmp_mnt) from paths for remote simulation.

3.13.2: Troubleshooting

Remote simulation attempts to diagnose common problems when a simulator is started. Two remote shell routines are performed to check the following five items, if any are not correct the simulator is killed and an error message output to explain what is required.

- **Unknown host** - Check if remote host name is entered correctly in the **Simulator Properties** popup or on command line. If so, contact your System Administrator as you are unable to access the required remote host.
- **.rhost file Error** - For remote simulation, the name of the local host and your username must be entered into the .rhost file located in your home directory. To continue, either add the line <hostname> <username> to your .rhost file or contact your System Administrator to make the required changes.
- **CWD Mount Error** - For remote simulation, the current working directory must be mounted for the remote machine. To continue, either change the current working directory by loading your input file from a directory mounted for both machines or contact your System Administrator to ensure the present directory is mounted on the remote host.
- **Write Permisson Error** - For remote simulation, write access must be available for the temporary directory. To continue, either change **Remote Tmp Directory** setting under the **Main Control** popup **Options** category to an accessible directory or contact your System Administrator to set the current remote tmp directory to be write accessible.
- **Tmp Dir Mount Error** - For remote simulation, the temporary directory must be mounted for both the local and remote machines. To continue, either change **Remote Tmp Directory** setting under the **Main Control** popup **Options** category to a directory mounted for both machines or contact your System Administrator to ensure the current remote tmp directory is mounted on the remote host in addition to the local machine.

Note: Use of remote simulation is not recommended and not supported when `DECKBUILD` is executed from a remote machine and displayed locally. Either remote login to a machine and execute `DECKBUILD` and the simulators on that host or run `DECKBUILD` on your local machine and use remote simulators.

This page is intentionally left blank.

4.1: Overview

This section contains a complete description of every statement and parameter used by DECKBUILD. The following information is provided for each statement:

- The statement name
- The syntax of the statement with a list of all the parameters of the statement and their type
- A description of each parameter
- An example of the correct usage of each statement

4.1.1: DeckBuild Commands

The following list identifies the commands that DECKBUILD executes. Each of these commands is described in subsequent sections:

- ASSIGN
- AUTOELECTRODE
- DEFINE
- ELSE
- EXTRACT
- GO
- IF
- IF .END
- L .END
- L .MODIFY
- LOOP
- MASK
- MASKVIEWS
- SET
- SOURCE
- STMT
- SYSTEM
- TONYPLOT
- UNDEFINE

4.2: ASSIGN

Provides a much richer version of the functionality provided by the existing SET statement (see Section 4.11: "SET").

Syntax

This is the syntax of the ASSIGN statement:

```
assign  name = <variable>          [print]

      (n.value = <expr_array> [delta=<expr> | ratio=<expr>] |
      1.value = <expr_array> |
      c.value = <qstring>          [delta=<expr>] |
      <c_array>
      )

      [level = <expr>]
```

with the following subsidiary definitions :

```
<expr_array> ->  <expr> |
                  (<expr>, <expr_array>) |
                  (<expr> <expr_array>)

<c_array>     ->  c<integer>=<qstring> |
                  c<integer>=<qstring> <c_array>
```

Description

The ASSIGN statement allows you to assign either a numerical (n), a logical (1) or a character (c) value to a variable. Numerical values may be arbitrary arithmetical expressions and may incorporate any of the standard functions mentioned in Section 4.11: "SET". All user-defined variables will be substituted before the expression is evaluated.

Arbitrarily, many variables may be assigned in the same deck.

Logical values may also be arbitrary numerical expressions. If any expression evaluates to a non-zero value, it is interpreted as true. Otherwise, it is interpreted as false. You can use the actual words "true" and "false". You can also assign arbitrary boolean expressions to logical values. The following operators are recognized:

```
logical AND      &

logical OR       |

logical NOT      ^
```

The usual relational operators are also recognised (>, <, >=, <=) with a single '=' character for the equals operator and the token ^= for the not-equals operator.

Note: Although unquoted strings are supported, you should **always** use quoted strings for character values for the sake of clarity.

You can assign a whole array of values to a variable. Arrays of numerical and logical arrays are written in the following manner:

```
(1, 2, 4, 8)
```

but arrays of character variables are written like this :

```
c2 = "Mary" c3 = "had" c5 = "a" c7 = "little" c11 = "lamb."
```

You can have many terms in a character array with their defining integers (the ones prefixed with 'c' for 'character') and not be sequential.

The array will be sorted in the increasing order of its defining integers.

Arrays are usually assigned to variables in loops. After each loop, the next value in the array will be assigned to the variable. If the end of the array comes before the end of the loop, the variable will revert to the first value in the array on the next pass.

You can also use the delta and ratio clauses to alter a variable on each pass through a loop. If you specify delta, that value will be added to the variable on each pass. If you specify ratio, the variable will be multiplied by that value on each pass.

If you specify an array of values, you cannot then specify either the delta or the ratio clauses.

You can specify a delta clause for a character value. This increment must be an integer and will be truncated if it isn't. This is an odd concept but is useful when, for example, you want to use a new output file on each iteration of a loop. A few examples will illustrate the idea. If the character value is a00 and delta is 4, then the first few values the variable takes will be a00, a04, a08, a12 and so on. Eventually, you will reach the values a92, a96, b00, b04, and so on. Incrementing lower-case 'z' by one produces lower-case 'a' but not upper-case 'A' and vice versa. You can also specify a negative delta with the obvious results.

An ASSIGN will persist until you encounter a second ASSIGN with the same variable name. If this happens, the old ASSIGN will be discarded and replaced by the new one. If an ASSIGN is outside of all loops, then the value of its variable never changes. If it's inside a loop, then its variable changes every time a new iteration of the loop begins.

If you specify the print keyword, the current value of the variable will print when initialized and will change each time thereafter.

You can use the level clause to have the value of the variable change when a particular member of a set of nested loops begins a new iteration. If the level you specify is positive, the loop is obtained by counting downwards from the zero level, the one outside of all loops. If the level is negative, the loop is obtained by counting upwards from the current level towards the outermost loop. So, level=-2 means change when the loop two above the present one starts a new iteration. level=2 means change when the next-to-the-outermost loop begins a new iteration.

As already mentioned, user-defined variables will be substituted before attempting expression evaluation. These variables are defined using the SET and ASSIGN statements. You can indicate the presence of a user-defined variable by prefixing it with '\$' or '@' or by surrounding it with braces like this:

```
${my_variable_1}, @{my_variable_2}.
```

Variables embedded withing quoted strings will be correctly substituted. "Bare" variables will be recognized provided they are surrounded by both spaces and parentheses. This usage, however, is very confusing and highly inadvisable.

Examples

1. In this example, param1 will take the values 1, 2 and 3 on the three passes through the loop.

```
loop steps=3
  assign name=param1 print n.value = 1 delta = 1
l.end
```

2. This generates the sequence aa.20, aa.16, aa.12, aa.08, aa.04 and aa.00 for param2.

```
loop steps=6 print
  assign name=param2 c.value = "aa.20" delta = -4
l.end
```

3. Followed by, "Mary", "had", "a", "little" and "lamb".

```
loop steps=5 print
  assign name=param3 c10="lamb." c3="Mary" c8="little" c4="had" c7="a"
l.end
```

In the two preceding examples, the double quotation marks will **not** be included when param2 and param3 are substituted into later expressions.

4. param1 takes the values 42, 38, 17, 42, 38.

```
loop steps=5 print
  assign name=param1 n.value = (42, 38, 17)
l.end
```

5. param1 takes the values 42, 45.2, 48.4, 51.6, 54.8.

```
loop steps=5 print
  assign name=param1 n.value = 42 delta = 3.2
l.end
```

6. param1 takes the values 42, 134.4, 430.08, 1376.26, 4404.02.

```
loop steps=5 print
  assign name=param1 n.value = 42 ratio = 3.2
l.end
```

7. This is a simple example illustrating the use of boolean expressions.

```
assign name=condition l.value = ($x > 0.0 & $y < 3.0)
```

If x and y represent coordinates, the value of condition will be true or false accordingly as the coordinates are in a required area of the structure. The value of \$condition could then be used as input to an IF statement.

8. It is worth emphasizing that ASSIGN can be used for the simplest of cases. See the following example:

```
assign name=e_charge n.value=1.6e-19
```

4.3: AUTOELECTRODE

Defines layout-based electrodes

Syntax

```
autoelectrode
```

Description

The `autoelectrode` command causes DECKBUILD to submit electrode definition statements to the current simulator. The electrode name and positioning information will be taken from the MASKVIEWS layout data.

Note: DECKBUILD only remembers the electrodes specified within each mask. Therefore, an `autoelectrode` statement must be used for every mask layer where electrodes are defined. This defines multiple electrodes for a single `autoelectrode` statement within the current mask.

See

Section 3.9: “IC Layout Interface”.

4.4: DEFINE and UNDEFINE

DEFINE replaces all subsequent occurrences of an identifier with a specified string.

UNDEFINE cancels this action.

Syntax

```
define    <identifier> <rest_of_line>
undefine  <identifier>
```

Description

The identifier should either be a quoted string or a well-formed identifier. That is, one which begins with a letter or an underscore and continues with an arbitrary sequence of letters, digits, underscores and periods.

Every time this token is identified thereafter, it will be replaced by the whole of the rest of the DEFINE statement from the end of the token down to the end of the line. This <rest_of_line> component may consist of any characters whatsoever.

You don't have to flag the presence of the defined (DEFINE) token using a '\$' or '@' prefix or any of the other methods mentioned in Section 4.2: "ASSIGN".

Substitution of a defined (DEFINE) token will persist until you encounter an UNDEFINE statement referencing the same token.

Substitution of defined (DEFINE) tokens will occur before each line is executed, unless the line begins with a % character. This also holds for the DEFINE and UNDEFINE lines themselves and has an odd corollary, which you can see in the examples section.

Examples

1. Here is a straightforward example:

```
define mypath /home/john_smith/tmp/logs
.
.
.
log outf=mypath/file1.log
.
.
.
log outf=mypath/file2.log
```

This pathology will define black as white.

```
define color black
.
.
.
define color white
```

To get the behavior you probably had in mind, do this :

```
define color black
```

```

.
.
.
%define color white

```

2. Something similar happens with the UNDEFINE command. In the next example, "black" is substituted for "color" in the UNDEFINE command and a no-op results.

```

define color black
.
.
.
undefine color

```

3. For an UNDEFINE to take effect, always use the '%' prefix. For example:

```

define color black
.
.
.
%undefine color

```

4.5: EXTRACT

Extracts information from the current simulation

Syntax

```
extract extract-parameters
```

Description

The `extract` statement is used to extract interesting information from the current simulation. See Chapter 5: “Extract” for a complete description.

See

Chapter 5: “Extract”.

4.6: GO

Interface between simulators

Syntax

```
go <simulator> [inflags=<> | outflags=<> | simflags=<> | cutline=<>|noauto]
```

Description

The GO statement tells DECKBUILD to shut down the current simulator and start up the specified simulator when the statement is executed. It is used to auto-interface between simulators.

simulator can be ssuprem3, athena, sminimos4, atlas, devedit, utmost, neurofab.

inflags specifies new load command flags for autointerface.

outflags specifies new save command flags for autointerface.

simflags specifies flags to be appended to default simulator argument.

cutline specifies a MASKVIEWS cutline file to be loaded into DECKBUILD.

noauto specifies that no autointerface occurs for this go statement.

Examples

If the current simulator is SSUPREM3, then this statement causes DECKBUILD to quit SSUPREM3 and start up ATHENA.

```
go athena
```

This will replace the default flags used in ATHENA auto interface command with "master" when loading and "flip.y" when saving.

```
go athena inflags=master outflags=flip.y
```

Note: One or more flags can be specified on the go line.

This statement will append "-V 2.2.1.R" to the default DEVEDIT argument to start version 2.2.1.R of the tool.

```
go devedit simflags="-V 2.2.1.R"
```

Note: Quotes are required where spaces used in flags or multiple flags used.

This loads the MASKVIEWS cutline default.sec from the specified directory into DECKBUILD.

```
go athena cutline="/usr/jdoe/default.sec"
```

This removes the currently loaded MASKVIEWS cutline.

```
go athena cutline=none
```

Note: The cutline flag should never be used with VWF.

The cutline flag cannot be used within VWF because is no guarantee that the specified directory path for the cutline file will exist on any of the remote machines in a network that VWF jobs can be sent to.

If the current simulator is ATHENA, then the following statement causes DECKBUILD to quit ATHENA and start up ATLAS but no autointerface between the two simulators will occur.

```
go atlas noauto
```

See

Section 3.8: “Auto Interfacing”

4.7: IF, ELSE and IF.END

These three commands together provide the standard IF block functionality.

Syntax

```
if cond = (<boolean_expr>)  
  else [cond = (<boolean_expr>)]  
  if.end
```

Description

The IF command starts the block. If its condition evaluates to true, then statements down to the next ELSE or IF.END line will be executed. If the condition evaluates to false, then there will be a search for an ELSE IF line whose condition evaluates to true. If you find such a line, the lines in its sub-block will be executed. At most, one sub-block in a given IF block will be executed.

The <boolean_expression> can be an arbitrary combination of boolean variables concatenated with AND, OR or NOT operators as described in Section 4.2: "ASSIGN".

You can nest IF blocks with each other and with LOOPS. As usual, an ELSE or an IF.END is associated with the most recent IF. There is no mechanism for using brackets or braces to enforce a particular nesting.

Example

```
if cond = (@MOSTYPE = "PMOS")  
  method gummel carriers = 1 holes  
  
  else  
    method gummel carriers = 1 electrons  
  
  if.end
```

4.8: LOOP, L.END and L.MODIFY

These three commands together provide the standard looping functionality.

Syntax

```
loop          steps = <expr> [print]
```

```
l.end         [break]
```

```
l.modify      [level = <expr>] [steps = <expr>] [next | break] [print]
```

Description

Every `loop` statement must have a corresponding `l.end` statement. All the commands between these two statements are executed repeatedly for the number of times given in the `steps` clause of the `loop` command. If you specify the `print` keyword, the values of all user-defined variables that vary under the control of the loop will print every time they change. If you specify the `break` keyword in the `l.end` statement, the loop will exit on its first iteration regardless of the value of `steps`.

Example: Simple Loop In DeckBuild and ATLAS

This example creates a simple resistor in ATLAS and uses the loop functionality in DECKBUILD to run two voltage solutions at 0.1V and 0.2V in ATLAS.

```
go atlas

mesh

x.m l=0.0 s=0.01
x.m l=0.1 s=0.01

y.m l=0.0 s=0.01
y.m l=0.1 s=0.01

region num=1 silicon

electrode name=top top

electrode name=bottom bottom

doping num=1 conc=1e17 n.type uniform

solve init
solve previous
```

```
set a=0.1
loop steps = 2

solve v1=$a
set a=$a*2

l.end

quit
```

Loops can of course be nested with each other and with IF blocks. When an l.end statement is encountered, it is associated with the most recent loop statement.

The l.modify statement changes the behavior of the current loop or one within which it is nested. You specify the level of the loop you wish to modify using the level clause, which is described in Section 4.2: “ASSIGN”. Without this clause, the current loop is assumed. You use the steps clause to change the number of times the loop will be executed. A value less than or equal to the current loop iteration count is acceptable and simply results in the loop exiting at the end of the current iteration.

The break keyword causes the loop to exit immediately.

The next keyword causes the loop to abandon the current iteration and to begin the next without executing any statements between the l.modify and the relevant l.end statements.

The print command switches on the printing of user-defined variables as described above.

Example

```
loop steps=3

    assign name=param1 print n.value = 1 delta = 1

loop steps=3
    assign name=param2 print n.value = 1 delta = 1
l.end

l.end
```

4.9: MASK

Defines the position of the process flow where photoresist or barrier material is added with the use of the MASKVIEWS IC layout interface.

Syntax

```
mask name="maskname"[misalign=<misalignment>/  
|bias=<bias>|delta_cd= <delta_cd>/  
|shrink=<shrink>|reverse|optolith]
```

Description

Mask is used to interface to SILVACO's general purpose layout editor MASKVIEWS. The mask statement defines the location where photoresist is deposited in the flow of processing events. The etched pattern is dependent on the MASKVIEWS cutline file, which must be loaded into DECKBUILD.

Name specifies the name of the layer that defines the photoresist patterning. This name must correspond to a mask level name contained in the MASKVIEWS cutline file loaded into DECKBUILD.

Bias and delta_cd increase or decrease the width of the deposited mask. For positive masks, a positive delta.bias decreases the etched hole(s) in the mask.

Misalignment shifts the entire specified mask left and right. Negative misalignment values shift the mask left, positive values right.

Shrink reduces the size of the specified layer by the ratio specified.

Reverse specifies that the mask polarity should be reversed or that negative type photoresist should be modeled.

Optolith specifies that the loaded MASKVIEWS cutline is from an **Optolith** layout. Therefore, **Optolith** syntax (layout commands) is used to define the photoresist pattern.

Examples

The delta value can be used to vary the Critical Dimension (CD) of the specified layer. The value operates on an edge-by-edge basis. For example, for an IC layout with a 1.0, micron wide "poly" the statement:

```
mask name="poly" delta=-0.1
```

creates a drawn poly length of 0.8 microns, meaning that 0.1 have been removed from each poly edge.

The bias command option performs the same operation as the delta command. This can be used globally to edit the bias of each layer. The bias command can be used with delta, such that the real value for CD reduction is the sum of the delta and bias values, per edge. For example, if an IC layout with 1.2 micron CD's is streamed-in from GDS2, and the final etch, then the final etch profile is known to be 0.9 microns due to a combination of biasing, photo-exposure, and over etch, then the offset is required to be constant. This is where the bias command can be used.

```
mask name="poly" bias=-0.15
```

In other words, 1.2 microns-0.9 microns=-0.3 microns =2(-0.15) microns, or -0.15 microns per edge.

Further experimentation might be required in addition to the fixed bias. This is where the delta command can be used. In this example:

```
mask name="poly" bias=-0.15 bias=-0.15 delta=-0.025
```

This simulates a true experiment in terms of CD variation.

The `misalign` command is used to offset a layer with respect to other layers. For example

```
mask name="poly" bias=-0.15 misalign=-0.1
```

causes the poly layer to be offset to the left by 0.1 microns.

The `shrink` command is used to reduce the size of all edges in the specified layer. For example, the statement below will reduce the layer edges by 50 percent.

```
mask name="poly" shrink=0.5
```

Misalignment and CD Experimentation

It is often necessary to experiment with either misalignment or the CDs of a layer. The MASKVIEWS-DECKBUILD interface supports this level of experimentation. DECKBUILD can be used to experiment with the cutline generated by MASKVIEWS. Each `mask` statement can be used to alter the cutline. The underlying mesh used by ATHENA is not changed with mask experimentation commands. VWF can be used to split on these values to generate RSM's relating to mask experimentation.

4.10: MASKVIEWS

Plots a layout file

Syntax

```
maskviews <layout file>
```

Description

This statement starts the MASKVIEWS layout editor and load the supplied layout file. If no layout file is specified, MASKVIEWS is invoked with no data.

Examples

This statement plots a layout file (which should be in the current directory).

```
maskviews layout.lay
```

See

MASKVIEWS USER'S MANUAL

4.11: SET

Sets the value of a user-defined variable or clear all existing variables.

Syntax

```
set <variable> = <value> | <expr> | <built_in_func> [nominal]
set clear
<built_in_func> = max    (<expr>, <expr>) |
                    min    (<expr>, <expr>) |
                    ave    (<expr>, <expr>) |
                    sin    (<expr>)          |
                    cos    (<expr>)          |
                    tan    (<expr>)          |
                    asin   (<expr>)          |
                    acos   (<expr>)          |
                    atan   (<expr>)          |
                    atan2  (<expr>, <expr>) |
                    sinh   (<expr>)          |
                    cosh   (<expr>)          |
                    tanh   (<expr>)          |
                    exp    (<expr>)          |
                    log    (<expr>)          |
                    log10  (<expr>)          |
                    pow    (<expr>, <expr>) |
                    sqrt   (<expr>)          |
                    ceil   (<expr>)          |
                    floor  (<expr>)          |
                    abs    (<expr>)          |
                    ldexp  (<expr>, <expr>) |
                    fmod   (<expr>, <expr>)
```

Description

The `set` command is used to set the value of a user-defined variable. The value can later be substituted using `$`-substitution, which replaces the variable name with its value when the variable is preceded by a dollar sign `'$'` or by the at sign `'@'`.

variable is a user-defined variable name. It may contain spaces or other non-alphabetical characters if it is delimited by double quotation marks..

<expr> is an algebraic expression consisting of numeric constants, `$`-substituted variables, algebraic operators (+, -, *, /, ^), and or the built-in functions shown.

`set` commands can be used in conjunction with extracted values. If a `$`-variable is to be substituted and if an existing `DECKBUILD` variable cannot be found, it is assumed to be a user-defined environment variable.

Synonyms for SET

We have introduced synonyms for the SET syntax to provide improved compatability with other products. Both of the following syntaxes are valid.

```
Assign|assign NAME=<variable> N.VALUE= <value> | <expr> | <built_in_func>
define <variable> <value> | <expr> | <built_in_func>
```

These will assign the chosen value or expression to the variable in the same way as the existing SET syntax. To reiterate, you can substitute the value later using \$-substitution, which replaces the variable name with its value when the variable is preceded either by a dollar sign '\$', or by the at sign '@'.

Examples

These statements show how to set variables and how to substitute with each other and in simulator syntax.

```
set time=30
set temp=1000
set press=1.0
set env="nitro"
set pi=3.1415
set "pi*2" = 2*$pi

diffuse time=$time temp=$temp press=$press $env hcl="$pi*2"
```

The following statements extract the thickness of the top layer of oxide in a structure and etch back that thickness plus 0.05 micron.

```
extract name="oxide thickness" thickness oxide
set etch_thickness = ($"oxide thickness"*10000) + 0.05

etch oxide dry thickness=$etch_thickness
```

Note: The thickness is measured in angstroms, so it is converted to microns first.

Variable names that contain spaces (generated by extract statements) must be quoted for \$-substitution, and the '\$' must precede the quoted string as shown.

For variable names with no spaces, quotation marks are optional.

The following statement will remove all existing variables.

```
set clear
```

The statements below show the use of the nominal flag.

```
extract name="oxide thickness" "oxide thickness" thickness_bad_syntax oxide
set "oxide thickness" = 0.5 nominal
etch oxide dry thickness=$oxide thickness
```

For this example, if the `extract` statement was successful, the value of "oxide thickness" would be set. Therefore, the `nominal set` statement would be ignored. But the `extract` syntax is incorrect, so the `extract` statement never creates the result variable and "oxide thickness" is set by the `nominal set` statement to 0.5.

4.12: SOURCE

Enables simulation commands to be executed from an external file

Syntax

```
SOURCE file
```

Description

The `SOURCE` statement enables simulation commands to be executed from an external file. The named file is read and placed in DECKBUILD's input buffer and is executed as if it were part of the input deck.

`file` is the name of a file that contains any valid simulator syntax or DECKBUILD statements, such as `extract` and `set`. The sourced file may source other files. If the file name does not begin with `'/'`, then it is assumed to be in the current directory.

Examples

The file to be sourced may contain part of an input deck including commands from any simulator. The following input deck fragment will perform a diffusion, access the file `include_file` for further commands, then revert back to the deposition step.

```
etch oxide all
#
# Source an external file
# Return to the input deck
implant bf2 dose=1.0e12 energy=35 pearson
include_file contains these statements:
#gate oxide grown here:-
diffus time=10 temp=900 dryo2 press=1.00 hcl%=3
```

The runtime output from this fragment will appear as:

```
ATHENA> etch oxide all
ATHENA> #
ATHENA> # Source an external file
ATHENA> source include_file
ATHENA> #gate oxide grown here:-
ATHENA> diffus time=10 temp=900 dryo2 press=1.00 hcl%=3
Solving time(sec.)      0 +      0.01      100%, np 106
Solving time(sec.)      0.01 +   0.173987   1739.87%, np 106
Solving time(sec.)   0.183987 +   0.187665   107.861%, np 106
*
Solving time(sec.)   0.371653 +   0.628347   334.823%, np 106
Solving time(sec.)      1 +      0.1      15.9148%, np 106
Solving time(sec.)      1.1 +    3.1396    3139.6%, np 106
Solving time(sec.)      4.2396 +   19.1813   610.947%, np 106
Solving time(sec.)     23.4209 +   93.4041   486.955%, np 106
```

```
Solving time(sec.)  116.825 +      150  160.593%, np 104
Solving time(sec.)  266.825 +      150    100%, np 104
Solving time(sec.)  416.825 +      150    100%, np 104
Solving time(sec.)  566.825 +   33.1751  22.1167%, np 104
ATHENA ># Return to the input deck
ATHENA> implant bf2 dose=1.0e12 energy=35 pearson
```

4.13: STMT

Enables you to define variables that change under the control of loops.

Syntax

```
stmt <parameters>
```

Where

```
<parameters> -> <parameter> | <parameter> <parameters>
```

```
<parameter> -> <variable> = <initial> [ : [ + | * ] <change> [ : <level> ] ]
```

That is, a `stmt` command must carry at least one parameter and may carry many (independent) parameters.

Description

This is effectively a shorthand for part of the `ASSIGN` statement.

The `<initial>`, `<change>` and `<level>` terms are all numerical expressions.

The value of the variable is re-evaluated **every** time the `STMT` command is encountered. If no arithmetical operator is specified or if the '+' sign appears explicitly, then addition is understood and the variable is re-evaluated as

```
<initial> + <change> * (count - 1)
```

where `count` is the current iteration count of the loop with level `<level>`.

If the multiplication operator ('*') appears, the variable is re-evaluated as

```
<initial> * pow(<change>, (count - 1))
```

where `pow` is the usual exponentiation function.

The `<change>` term defaults to 0 in the addition case and 1 in the multiplication case. The `<level>` term defaults to the current loop level. This means that if you only specify `<initial>`, the variable will be a constant.

Examples

1. In this example `param1` will take the values 1, 2, 3, 4 and 5.

```
loop steps=5 print
  stmt param1=1:1
1.end
```

2. In this example `param1` will take the values 1, 2, 4, 8 and 16.

```
loop steps=5 print
  stmt param1=1:*2
1.end
```

4.14: SYSTEM

Allows DECKBUILD to execute UNIX system commands within a simulation deck.

Syntax

```
SYSTEM <UNIX command>
```

Description

The SYSTEM command allows you to execute shell scripts or perform other UNIX tasks directly from the simulation deck. The command is blocking, meaning that the simulation does not continue until the SYSTEM command has finished execution.

To use this feature, enable the SYSTEM commands in the Main Control Options Popup (see Figure 3-6). To enable system commands for VWF AUTOMATION TOOLS, set the environment variable DB_SYSTEM_OPTION to any value.

Examples

```
system rm history*.str
```

Note: Redirection of the system command output (i.e., `system ls * .in > file.out`) cannot be achieved as the output is already redirected by DECKBUILD.

4.15: TONYPLOT

Plots a file

Syntax

```
tonyplot -args
```

Description

This statement causes DECKBUILD to save a temporary file from the current simulator and start up TONYPLOT with that file loaded. The temporary file is removed when TONYPLOT exits.

-args, if specified, are passed directly to TONYPLOT (as if invoked from the command line). If any of -st, -da, or -over and a file name is specified, DECKBUILD uses the named file instead of saving and plotting the current structure.

DECKBUILD also detects if the structure to be plotted is 3D and use TONYPLOT3D if required.

Examples

This statement saves the current file and starts TONYPLOT.

```
tonyplot
```

This statement plots a file (which should be in the current directory).

```
tonyplot -st well.str
```

See

Section 3.3: "Main Control".

5.1: Overview

DECKBUILD has a built-in extraction language that allows measurement of physical and electrical properties in a simulated device. The result of all extract expressions is either a single value (such as X_j for process or V_t for device), or a two-dimensional curve (such as concentration versus depth for process or gate voltage versus drain current for device).

EXTRACT forms a “function calculator” that allows you to combine and manipulate values or entire curves quickly and easily. You can create your own, customized expressions, or choose from a number of standard routines provided for the process and device simulators. You can take one of the standard expressions and modify it as appropriate to suit your needs. EXTRACT also has variable substitution capability so that you can use the results of previous extract commands.

EXTRACT has two built-in 1D device simulators, QUICKMOS and QUICKBIP, for specialized cases of MOS and bipolar electrical measurement. Both QUICKMOS and QUICKBIP run directly from the results of process simulation for fast, easy and accurate device simulation.

5.2: Process Extraction

DECKBUILD's process extraction window is shown below (Figure 5-1).

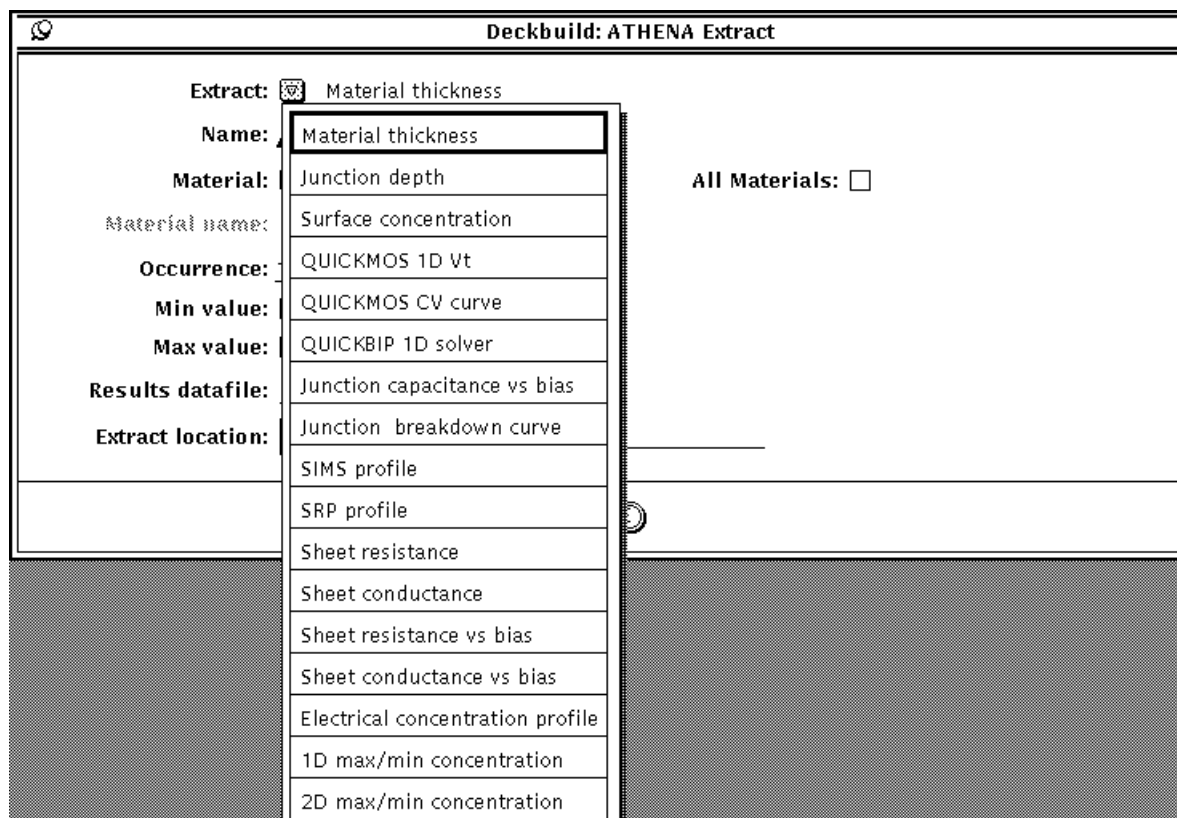


Figure 5-1: Process Extraction

You may use this window to look at the following:

- **Material thickness** measures the thickness of the nth occurrence of any material or all materials in the structure.
- **Junction depth** measures the depth of any junction occurrence in the nth occurrence of any material.
- **Surface concentration** measures the surface concentration of any dopant, or net dopant, in the nth occurrence of any material.
- **QUICKMOS 1D Vt** calculates the one-dimensional threshold voltage of a MOS cross section using the built-in QUICKMOS 1D device simulator. The gate voltage range defaults between 0 to 5 Volts but can be specified as required. The substrate can also be fixed at any bias. Qss and device temperature values may also be specified.
- **QUICKMOS CV curve** creates a CV curve of a MOS cross section using QUICKMOS. This shows capacitance as a function of either gate voltage or substrate voltage with the other terminal held at any fixed bias. Qss and device temperature values may also be specified.
- **QUICKBIP 1D solver** measures any of 22 BJT Gummel-Poon parameters, plus any forward or reverse IV curve. See the Section 5.9: "QUICKBIP Bipolar Extract" for more information and examples.
- **Junction capacitance versus bias** calculates the junction capacitance of a specified p-n junction within any region as a function of applied bias to that region. Qss and device temperature values can also be specified.

- **Junction breakdown curve** calculates the electron or hole ionization integral of any region as a function of applied bias to that region. This calculation uses the Selberherr impact ionization model. (see “Impact” command section and “Impact Ionization” physics section within the ATLAS manual). You can modify the Selberherr model default values and specify Qss and device temperature values.
- **SIMS profile** calculates the concentration profile of a dopant in a material layer.
- **SRP profile** calculates the SRP (Spreading Resistance Profile) in a silicon layer.
- **Sheet resistance and sheet conductance** calculates the sheet resistance or conductance of any p-n region in any layer in an arbitrary structure. You can specify the bias of any region in any layer, the Qss of any material interface and the device temperature. A flag for carrier freezeout calculations can also be set (see “Incomplete Ionization Of Impurities” physics section within the ATLAS manual).
- **Sheet resistance and sheet conductance versus bias** calculates the sheet resistance or conductance of one or more regions as a function of applied bias to any region. Qss and device temperature values can also be specified.
- **Electrical concentration profiles** measures electrical distributions versus depth. You can also specify the bias of any region in any layer and the Qss of any material interface. The device temperature can also be set to the required value. The following distributions are calculated:
 - electrons
 - holes
 - electron quasi-fermi level
 - hole quasi-fermi level
 - intrinsic concentration
 - potential
 - electron mobility
 - hole mobility
 - electric field
 - conductivity
- **1D maximum/minimum concentration** measures the peak or minimum concentration of any dopant or net dopant, for a specified 1D cutline, in the nth occurrence of any material or all materials, and also within any junction-defined.
- **2D maximum/minimum concentration** measures the peak or minimum concentration of any dopant or net dopant, for the whole 2D structure or within a specified area, in any material or all materials, and also within any junction-defined. The actual xy coordinates of the maximum or minimum concentration can also be retrieved.
- **2D material region boundary** returns the maximum or minimum boundary of the selected material region for either X or Y axis. Therefore, the outer boundaries of any material region can be extracted.
- **2D concentration area** integrates specified concentration of any dopant or net dopant for whole 2d structure or within a specified location.
- **2D maximum concentration file (CCD)** creates a **Data Format** file with the XY coordinates and the actual values of the maximum concentrations stepping across the structure. This file can be loaded into TONYPLOT when in **-ccd** mode to show a line of maximum concentration across a device.
- **ED tree** creates one branch of a **Smile** plot or **ED** tree from multiple **Defocus** distance against **Critical Dimension (CD)** plots created for a sweep of **Dose** values by OPTOLITH. These plots are all written in a single Data format file.
- **Elapsed time** extracts time stamps from a specified start time at any point in a simulation. You can reset the start time as required.

Note: This extraction is not CPU time.

The built-in 1D Poisson device simulator is used to calculate sheet resistance and conductance and the electrical concentration profiles.

With the exception of 2D extractions, all the process extraction routines are available from both 1D and 2D process simulators. In the case of the 2D simulators, a cross section x or y value or region name (used in conjunction with MASKVIEWS) determines the 1D section to use.

Note: An error will be returned for attempted extractions on 3D structure files.

5.2.1: Entering a Process Extraction Statement

To place an `extract` statement in your process deck, select **Commands**→**Extract...** The **Extraction** popup appears. The popup for ATHENA is shown in Figure 5-2.

The screenshot shows a window titled "Deckbuild: ATHENA Extract". Inside, there are several input fields and checkboxes. The "Extract:" dropdown is set to "Material thickness". The "Name:" field is empty. The "Material:" dropdown is set to "Silicon", and the "All Materials:" checkbox is unchecked. The "Material name:" field is empty. The "Occurrence:" field shows a range from 1 to 10 with a slider. The "Min value:" and "Max value:" fields are empty, each with a checkbox and a unit symbol (Å). The "Results datafile:" field is set to "results.final". The "Extract location:" field has three checkboxes: "X", "Y", and "Region name", with "X" and "Y" checked. The "Value:" field is empty. A "WRITE" button is located at the bottom center of the window.

Figure 5-2: The ATHENA Extraction Popup

Choose the extract routine you want by activating a choice on the **Extract** setting. The popup changes size and display different items, depending on which routine you choose. Then, enter or choose the desired information for each item on the popup. An extract name is always be required. Optionally, enter the minimum or maximum desired cutoff values by checking **Min value** or **Max value** and entering a value. By default, all extract results are written to a file named `results.final`. But using the **Results datafile** field allows you to specify the results file for each individual extract statement. Material and impurity names are selected using a **Chooser** (Figure 5-3).

Figure 5-3: Material Chooser Popup

If the required option is not present in the default setting, select the **User** filter to search for other materials/impurities. The **Hide Worksheet Result** setting specifies that this extract should not be displayed in the VWF INTERACTIVE TOOLS worksheet. This prevents extracts used for calculation purposes only from cluttering the worksheet results.

Finally, place the text caret at the desired point in the deck and click on the **WRITE** button. The extract syntax is written to the deck.

5.2.2: Extracting a Curve

Some of the process extraction statements create a two-dimensional curve as a result, rather than a single value. For instance, `extract` constructs a data set of concentration versus depth for the SIMS, SRP, and electrical quantities distributions. You can use the resulting 2-D curve for measurement and testing and as a target on the OPTIMIZER worksheet so that you can optimize against 2-D curves.

EXTRACT provides several additional options to 2-D curve support: axis layout, axis attributes, optional computation of area under the curve, and optional outfile. These options are the same regardless what type of curve (for instance, QUICKMOS CV and SIMS profile) you are extracting.

The ATHENA Extract popup showing the SIMS Profile is shown in Figure 5-4.

Deckbuild: ATHENA Extract

Extract: ☒ SIMS profile

Name: _____

Material: Silicon Material... All Materials: ☐

Material name: _____

Occurrence: 1 ☐ 10

Impurity: Net Doping Impurity...

X vs Y axis: Depth vs Concentration Concentration vs Depth Compute curve area: ☐

X axis attributes: log log10 abs sqrt

Y axis attributes: log log10 abs sqrt

X axis bounds: ☐

X axis start: _____ X axis end: _____

Store X/Y datafile: Yes No Filename: extract.dat

Results datafile: results.final Hide worksheet result: ☐

Extract location: X Y Region name Value: _____

WRITE

Figure 5-4: ATHENA Extract Popup with SIMS Profile

The following options are available:

- **X vs Y axis** determines the x and y axes of the resulting profile curve. The default (which should always be used unless you plan to customize the resulting extract expression) is that the x axis is depth into the material, and the y axis is the concentration.
- **Curve X axis bounds** specifies whether to create the curve for the whole X axis or for only a required section. If selected, **X axis value** fields become active, enter values in the same units as the resulting curve. This is useful for extracting local maxima and minima.
- **X axis attributes** and **Y axis attributes** allows you to modify the data values on each axis independently. To compute net concentration versus depth, you can select **abs** on the y axis (concentration), and select nothing on the x axis (depth). **abs** is always evaluated before taking the log or square root of the data.
- **Curve X axis bounds** specifies whether to create the curve for the whole X axis or for only a required section. If selected, **X axis value** fields become active, enter values in the same units as the resulting curve. This is useful for extracting local maxima and minima.
- **Store X/Y datafile** stores an output file in TONYPLOT data format if set to **Yes**. You can plot the data file in TONYPLOT using the `-da` option. You can also read the data file directly into the OPTIMIZER worksheet as a target if desired.
- **Compute curve area** computes the area under the curve. When checked, it causes several other items to become active:
- **Area X axis bounds** tells EXTRACT whether to integrate the area under the curve along its entire length or just for a bounded portion of the X axis. If you select **Bounded**, then **X axis start** and **X axis stop** become active. Enter **start** and **stop** values in the same units as the resulting curve.

To construct the 2-D curve, set each item on the popup in turn and click on **WRITE**.

Depth is always computed as distance from the top of the selected material layer and occurrence. Depth starts from 0 and increases through the material.

5.3: Customized Extract Statements

In addition to the simple curve primitives shown on the popup, you can edit the input deck directly to make customized curves. Examples include extracting maxima and minima on the curve, combining axes using a function definition, looking at slopes of tangent lines, intercepts of sloped lines. The EXTRACT syntax is described below, followed by examples of process extraction. See the examples listed under Section 5.4: “Device Extraction” for more information.

5.3.1: Extract Syntax

Text inside matching pairs of `/*` and `*/` delimiters are comments. These are used to clarify the meaning of the syntax and also as definitions for the most primitive types, such as `<QSTRING>`.

The backslash character (`\`) at the end of a line indicates a continuation line.

Many of the optional parameters (the ones enclosed in square brackets) have default values. Some of these defaults are given immediately after they appear. Others appear in more than one place and so are collected at the end.

Description

`<EXTRACT_STATEMENT>` :

`<EXTRACT_SINGLE_LINE_GENERAL>`

`<EXTRACT_MULTIPLE_LINE_GENERAL>`

`<EXTRACT_2D_MAX_MIN_CONC>`

`<EXTRACT_TIME>`

`<EXTRACT_SIMPLE>`

`<EXTRACT_SINGLE_LINE_GENERAL>` :

`[extract init infile=QSTRING]`

`/* In default of the above line, a temporary structure file representing the current state of the device will be constructed. */`

`extract [name=QSTRING] <EXTRACT_SINGLE_LINE_PARTICULAR> \`
`[datafile=QSTRING] [hide]`

`<EXTRACT_MULTIPLE_LINE_GENERAL>` :

`[extract init infile=QSTRING]`

`/* In default of the above line, a temporary structure file representing the current state of the device will be constructed. */`

`extract start <EXTRACT_MULTIPLE_LINE_SETUP_N>`

`[extract cont <EXTRACT_MULTIPLE_LINE_SETUP_N> ...]`

```
/* zero or more instances of the extract cont line may appear. */

extract done [name=<QSTRING>] <EXTRACT_MULTIPLE_LINE_DONE_N>          \
            [datafile=<QSTRING>] [hide]

/* There are five pairs of definitions for<EXTRACT_MULTIPLE_LINE_SETUP_N>
   and <EXTRACT_MULTIPLE_LINE_DONE_N>, with N replaced by 1, 2, 3, 4 or 5.
   Elements from different pairs (ones with different values of N) must NOT
   appear in the same statement. */

<EXTRACT_2D_MAX_MIN_CONC> :

[extract init infile=<QSTRING>]

/* In default of the above line, a temporary structure file representing the
   current state of the device will be constructed. */

extract [name=<QSTRING>] 2d.max.conc | 2d.min.conc [interpolate]      \
[<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>]                        \
[min.v=<EXPR>][max.v=<EXPR>]                                          \
[x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR> |                \
 y.max=<EXPR> y.min=<EXPR> region=<QSTRING>]                          \
[datafile=<QSTRING>] [hide]

[extract [x_pos_name=<QSTRING>] x.pos

extract [y_pos_name=<QSTRING>] y.pos]

/* x_pos_name and y_pos_name will default to the name in the main extract
   statement, with "X position" and "Y position" appended respectively. */

<EXTRACT_TIME> :

extract [name=<QSTRING>] clock.time [start_time=<EXPR>]             \
[datafile=<QSTRING>]

<EXTRACT_SIMPLE> :

extract [name=<QSTRING>] <EXPR> [datafile=<QSTRING>]

<EXTRACT_SINGLE_LINE_PARTICULAR> :

<CURVE_FUNC> (<CURVE_SINGLE_LINE>) [outfile=<QSTRING>]
```

```
<EXTRACT_MULTIPLE_LINE_DONE_5>
```

```
thickness [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>]
```

```
xj [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] [junc.occno=<EXPR>]
```

```
surf.conc [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] [<IMPURITY>]
```

```
ldvt [ptype | ntype] [min.v=<EXPR>] [max.v=<EXPR>] \
[bias=<EXPR>] [bias.start] [bias.stop=<EXPR>] [bias.step=<EXPR>] \
[vb=<EXPR>] [temp.val=expr] [soi] [qss=<EXPR>] [workfunc=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>]
```

```
/* Default values : ptype, vb=0.0, qss=0 */
```

```
max.conc | min.conc [<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \
[region.occno=<EXPR>]
```

```
/* region.occno will default to all regions. */
```

```
2d.conc.file [<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR>]
```

```
max.conc.file | min.conc.file [<IMPURITY>] [<MATERIAL>] [xstep=<EXPR>] \
[x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR>]
```

```
max.bound | min.bound x.val=<EXPR> | y.val=<EXPR> \
[min.v=<EXPR>] [max.v=<EXPR>] [MATERIAL] [mat.occno=<EXPR>]
```

```
max.bound | min.bound x.pos | y.pos xval=<EXPR> y.val=<EXPR> \
[MATERIAL] [min.v=<EXPR>] [max.v=<EXPR>]
```

```
2d.area [<IMPURITY>] [x.step=<EXPR>] [min.v=<EXPR>] [max.v=<EXPR>] \
[x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR> |
y.max=<EXPR> y.min=<EXPR> region=<QSTRING>]
```

```
/* Default value : x.step = 10% of device size */
```

<CURVE_FUNC> (<CURVE_ARG>) :

<CURVE_ARG>

min (<CURVE_ARG>)

/* Returns min y val for curve. */

max (<CURVE_ARG>)

/* Returns max y val for curve. */

ave (<CURVE_ARG>)

/* Returns average value for curve. */

slope | xintercept | yintercept (maxslope | minslope (<CURVE_ARG>))

/* Takes the tangent to the curve with either the least or the greatest
slope and returns either the slope of this tangent, or its x intercept,
or its y intercept. */

area from (<CURVE_ARG>) [where x.min=<EXPR> and x.max=<EXPR>]

/* Determines the area under the specified curve between the x limits
defined by the min and max expressions. */

x.val from (<CURVE_ARG>) where y.val=<EXPR> [and val.occno=<EXPR>]

y.val from (<CURVE_ARG>) where x.val=<EXPR> [and val.occno=<EXPR>]

/* Determines the x (or y) ordinate on the curve where the corresponding
y (or x) ordinate is equal to the constant expression for the occurrence
specified. Linear interpolation is used between points on the curve.*/

grad from (<CURVE_ARG>) where x.val=<EXPR> | y.val=<EXPR>

```

/* Determines the gradient at the first x (or y) ordinate on the curve where
   the corresponding y (or x) value is equal to the consent expression.
   Linear interpolation is used between points on the curve.*/

```

```

<CURVE_SINGLE_LINE> :

```

```

curve (<AXIS_FUNC> (bias),
      <AXIS_FUNC> (ldcapacitance [vg=<EXPR>] [vb=<EXPR>] [bias.ramp=vg|vb]\
                  [bias.step=<EXPR>] [bias.start=<EXPR>] \
                  [bias.stop=<EXPR>][temp.val=expr][soi][qss=<EXPR>] \
                  [workfunc=<EXPR>] \
                  [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] \
                  ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
      )

```

```

/* Default values : vg=0.0, vb=0.0, bias.ramp=vg, qss=0 */

```

```

curve (<AXIS_FUNC> (depth),
      <AXIS_FUNC> ([<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \
                  [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] \
                  ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
      )

```

```

curve (<AXIS_FUNC> (depth),
      <AXIS_FUNC> (srp \
                  [material="silicon"|"polysilicon"] [mat.occno=<EXPR>] \
                  [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] \
                  ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
      )

```

```

curve (<AXIS_FUNC> (<DEV_AXIS>),
      <AXIS_FUNC> (<DEV_AXIS>) \
      [, xmin=<EXPR> xmax=<EXPR>] \
      )

```

```

deriv (<AXIS_FUNC> (<DEV_AXIS>),
      <AXIS_FUNC> (<DEV_AXIS>) \
      [, <INTEGER>] \
      )

```

```

/* The integer is of course the nth derivative and its default value is 1.*/

```

```
edcurve (<DEFOCUS_AXIS>, <CRITICAL_DIMENSION_AXIS>, <DOSE_AXIS>, \
        dev=<EXPR> datum=<EXPR> x.step=<EXPR>)
```

<AXIS_FUNC> (<AXIS_ARG>) :

<AXIS_ARG>

<AXIS_ARG> + <EXPR>

<EXPR> + <AXIS_ARG>

<AXIS_ARG> + <AXIS_ARG>

<AXIS_ARG> - <EXPR>

<EXPR> - <AXIS_ARG>

<AXIS_ARG> - <AXIS_ARG>

<AXIS_ARG> / <EXPR>

<EXPR> / <AXIS_ARG>

<AXIS_ARG> / <AXIS_ARG>

<AXIS_ARG> * <EXPR>

<EXPR> * <AXIS_ARG>

<AXIS_ARG> * <AXIS_ARG>

<AXIS_ARG> ^ <EXPR>

<EXPR> ^ <AXIS_ARG>

<AXIS_ARG> ^ <AXIS_ARG>

-<AXIS_ARG>

abs (<AXIS_ARG>)

log (<AXIS_ARG>)

log10(<AXIS_ARG>)

sqrt (<AXIS_ARG>)

atan (<AXIS_ARG>)

<EXPR> :

<NUMBER>

\$variable | \$"variable"

/* deckbuild set variable, see section 5.8.2: Variable Substitution */

expr + expr

expr - expr

expr / expr

```

    expr * expr

    (expr)
    -expr

<EXTRACT_MULTIPLE_LINE_SETUP_1> :

    <EXTRACT_MULTIPLE_LINE_SETUP_A>

<EXTRACT_MULTIPLE_LINE_DONE_1> :

    <CURVE_FUNC> (<CURVE_MULTIPLE_LINE_1>) [outfile=<QSTRING>]

<CURVE_MULTIPLE_LINE_1> :

    curve (<AXIS_FUNC> (bias),
        <AXIS_FUNC> (ldjunc.cap [<MATERIAL>] [mat.occno=<EXPR>]
            [region.occno=<EXPR>] [junc.occno=<EXPR>]
            [temp.val=<EXPR>][soi][qss=<EXPR>][workfunc=<EXPR>]
            [y.val=<EXPR>|x.val=<EXPR>|region=<QSTRING>]
        )
        [, xmin=<EXPR> xmax=<EXPR>]
    )

<EXTRACT_MULTIPLE_LINE_SETUP_2> :

    <EXTRACT_MULTIPLE_LINE_SETUP_A>

<EXTRACT_MULTIPLE_LINE_DONE_2> :

    <CURVE_FUNC> (<CURVE_MULTIPLE_LINE_2>) [outfile=<QSTRING>]

<CURVE_MULTIPLE_LINE_2>:

    curve (<AXIS_FUNC> (bias),
        <AXIS_FUNC> ([p.ion | n.ion] [<MATERIAL>] [mat.occno=<EXPR>]
            [region.occno=<EXPR>] [junc.occno=<EXPR>]
            [temp.val=<EXPR>][soi][qss=<EXPR>][workfunc=<EXPR>]
            [y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>]
            [an1=<EXPR>] [an2=<EXPR>] [bn1=<EXPR>] [bn2=<EXPR>]
            [ap1=<EXPR>] [ap2=<EXPR>] [bp1=<EXPR>] [bp2=<EXPR>]
            [betan=<EXPR>][betap=<EXPR>][egran=<EXPR>]
        )
        [, xmin=<EXPR> xmax=<EXPR>]
    )

    /* Default value : p.ion

        an1=2.03e5, an2=7.03e5, bn1=1.231e6, bn2=1.231e6,
        ap1=6.71e5, ap2=1.582e6, bp1=1.693e6, bp2=2.036e6,

```

betan=1.0, betap=1.0, egran=4e5

See Appendix A5: Threshold Voltage Calculation. */

<EXTRACT_MULTIPLE_LINE_SETUP_3> :

<EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

<EXTRACT_MULTIPLE_LINE_DONE_3> :

<CURVE_FUNC> (<CURVE_MULTIPLE_LINE_3>) [outfile=<QSTRING>]

<CURVE_MULTIPLE_LINE_3> :

```
curve (<AXIS_FUNC> (bias),                                     \
      <AXIS_FUNC> (lds.sheet.res | ldp.sheet.res | ldn.sheet.res | \
                   ldconduct   | ldp.conduct   | ldn.conduct   | \
                   [material="silicon" | "polysilicon"]          \
                   [region.occno=<EXPR>] [mat.occno=<EXPR>]        \
                   [y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>] \
                   [workfunc=<EXPR>] [soi] [semi.poly] [incomplete] \
                   [temp.val=<EXPR>]                               \
      )                                                         \
      [, xmin=<EXPR> xmax=<EXPR>]                               \
    )
```

<EXTRACT_MULTIPLE_LINE_SETUP_4> :

<EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

<EXTRACT_MULTIPLE_LINE_DONE_4> :

<CURVE_FUNC> (<CURVE_MULTIPLE_LINE_4>) [outfile=<QSTRING>]

<CURVE_MULTIPLE_LINE_4> :

```
curve (<AXIS_FUNC> (bias),                                     \
      <AXIS_FUNC> (n.conc      | p.conc      | n.qfl      | p.qfl      | \
                   intrinsic | potential    | n.mobility | p.mobility | \
                   efield    | econductivity \
                   [material="silicon" | "polysilicon"]          \
                   [region.occno=<EXPR>] [mat.occno=<EXPR>]        \
                   [y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>] \
                   [workfunc=<EXPR>] [soi] [semi.poly] [temp.val=<EXPR>] \
      )                                                         \
      [, xmin=<EXPR> xmax=<EXPR>]                               \
    )
```

<EXTRACT_MULTIPLE_LINE_DONE_5> :

```

sheet.res|p.sheet.res|n.sheet.res|conduct|p.conduct|n.conduct      \
[material="silicon"|"polysilicon"] [region.occno=<EXPR>]           \
[mat.occno=<EXPR>]                                                  \
[y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>]                    \
[workfunc=<EXPR>] [soi] [semi.poly] [incomplete] [temp.val=<EXPR>]

<EXTRACT_MULTIPLE_LINE_SETUP_5> :

    <EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

<EXTRACT_MULTIPLE_LINE_SETUP_A> :

    [<MATERIAL>] [mat.occno=<EXPR>] [region.occno=<EXPR>]          \
    [bias=<EXPR>] [bias.start=<EXPR>] [bias.step=<EXPR>] [bias.stop=<EXPR>] \
    [y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>]

<EXTRACT_MULTIPLE_LINE_SETUP_B> :

    [interface.occno=<EXPR>] [qss=<EXPR>]

    /* Default value : qss=1e10 */

<DEV_AXIS> :

    v."<electrode>"          /* voltage at electrode */

    i."<electrode>"          /* current at electrode

    c."<electrode1>" "<electrode2>" /* capacitance between
                                   electrode1 and electrode2 */

    g."<electrode1>" "<electrode2>" /* conductance between
                                   electrode1 and electrode2 */

    vint."<electrode>"      /* internal voltage at electrode */

    time                  /* transient time */

    temperature | temp    /* device temperature */

    frequency | freq      /* frequency */

    beam."<beam no>"        /* light intensity for specified beam */

    s.imaginary."<Mode>"    /* imaginary component of specified "S" code*/

```

s.real."<Mode>"	/* real component of specified "S" code */
h.imaginary."<Mode>"	/* imaginary component of specified "H" code*/
h.real."<Mode>"	/* real component of specified "H" code */
ie."<electrode>"	/* electron current */
q."<electrode>"	/* charge */
id."<electrode>"	/* displacement current */
ireal."<electrode>"	/* real component of current */
iimag."<electrode>"	/* imaginary component of current */
ifn."<electrode>"	/* fowler nordhiem current */
ihe."<electrode>"	/* hot electron current */
ihh."<electrode>"	/* hot hole electron current */
wfd."<electrode>"	/* workfunction difference */
rl."<electrode>"	/* lumped resistance */
cl."<electrode>"	/* lumped capacitance */
ll."<electrode>"	/* lumped inductance */
vcct.node."<circuit node>"	/* circuit bias */
icct.node."<circuit node>"	/* circuit current */
rhoe."<layer>"	/* Electron sheet resistance for layer */
rhoh."<layer>"	/* Hole sheet resistance for layer */
rho."<layer>"	/* Total sheet resistance for layer */
vlayer."<layer>"	/* Bias on layer */
sm."<mode>"	/* Photon density mode */
pm."<mode>"	/* Laser power per mirror mode */
gm."<mode>"	/* Gain mode */
vcct.real."<circuit node>"	/* Real circuit bias */
vcct.imag."<circuit node>"	/* Imaginary circuit bias */
icct.real."<circuit node>"	/* Real circuit current */

```
icct.imag."<circuit node>"      /* Imaginary circuit current */
abcd.real."<mode>"               /* ABCD real parameter */
abcd.imag."<mode>"              /* ABCD imaginary parameter */
y.real."<mode>"                 /* Y real parameter */
y.imag."<mode>"                 /* Y imaginary parameter */
z.real."<mode>"                 /* Z real parameter */
z.imag."<mode>"                 /* Z imaginary parameter */
probe."<probe name>"           /* Atlas probe values */
elect."<PARAMETER>"            /* Value for specified electrical
                               parameter */
```

<PARAMETER> :

```
time
light frequency
freq
frequency
temp
temperature
current gain
unilateral power
gain frequency
max transducer power gain
luminescent power
luminescent wavelength
optical source frequency
available photo current
source photo current
optical wavelength
```

position xhole mobility
time step magnitude
time step number
total integration time
cutoff frequency
distance along line
norm intensity
integrated e- conc
integrated h+ conc
channel sheet conductance
photon energy
photon density gain
spontaneous emission rate
electron mobility
hole current generation rate
lattice temp
electric field
recombination rate
displacement current
electron conc
hole conc
electron temp
hole temp
relative permittivity
potential

<DEFOCUS_AXIS> :

da.value"DEFOCUS" | da.value"<CURVE_NUMBER>" "DEFOCUS"

<CRITICAL_DIMENSION_AXIS> :

da.value"Cds" | da.value"<CURVE_NUMBER>" "Cds"

<DOSE_AXIS> :

da.value"DOSE" | da.value"<CURVE_NUMBER>" "DOSE"

<CURVE_NUMBER> :

/* Integer specifying which curve when multiple curves are present in a
DA format file. */

<MATERIAL> :

Silicon
Oxide
SiO~2
Oxynitride
Nitride
Si~3N~4
Polysilicon
Photoresist
Barrier
Aluminum
Tungsten
Titanium
Platinum
Cobalt
Tungsten Silicide
Titanium Silicide
Platinum Silicide
Cobalt Silicide
GaAs
AlGaAs
InGaAs
SiGe
InP
6H-SiC
4H-SiC
3C-SiC
Germanium
material=<QSTRING>

<IMPURITY> :

Boron
Phosphorus
Arsenic
Bf2

```
Antimony
Silicon
Zinc
Selenium
Beryllium
Magnesium
Aluminum
Gallium
Carbon
Indium
Chromium
Germanium
impurity=<QSTRING>
```

<NUMBER> :

```
/* Real or integer value. */
```

<QSTRING> :

```
/* Quoted string, for example, "silicon". */
```

5.3.2: DEFAULTS

The following default values will be assumed.

name=<QSTRING>	: name=None
<MATERIAL>	: material="silicon"
<IMPURITY>	: impurity="net doping"
mat.occno=<EXPR>	: mat.occno=1
junc.occno=<EXPR>	: junction.occno=1
region.occno=<EXPR>	: region.occno=1
interface.occno=<EXPR>	: interface.occno=1
val.occno=<EXPR>	: val.occno=1
datafile=<QSTRING>	: datafile="results.final"
outfile=<QSTRING>	: outfile="extract.dat"
bias.step=<EXPR>	: bias.step=0.25
bias.start=<EXPR>	: bias.start=0.0

```

bias.stop=<EXPR>      : bias.stop=5.0

temp.val=<EXPR>        : temp.val=300.0

soi                   : FALSE

semi.poly             : FALSE

incomplete            : FALSE

x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING> : x.val is set to be 5% from
                                                left-hand side of structure.

```

5.3.3: Examples of Process Extraction

Note: You can enter `extract` commands on multiple lines using a backslash character for continuation. The syntax, however, shown below should be entered on a single line although shown on two or more lines.

The following examples assume to be extracting values from the current simulation running under DECKBUILD. You can use saved standard structure files directly with `extract` using the syntax below.

```
extract init infile="filename"
```

Material Thickness

Extract the thickness of the top (first) occurrence of Silicon Oxide for a 1D cutline taken where $Y=0.1$ (Assume 2D structure). A warning is then displayed if results cross boundaries set by `max.v` and `min.v`.

```
extract name="tox" thickness material="SiO~2" mat.occno=1 y.val=0.1 min.v=100
max.v=500
```

oxide can be substituted for the material="SiO~2".

Junction Depth

Extract the junction depth of the first junction occurrence in the top (first) occurrence of silicon for a 1D cutline taken where $X=0.1$.

```
extract name="j1 depth" xj material="Silicon" mat.occno=1 x.val=0.1 junc.occno=1
```

Surface Concentration

Extract the surface concentration net doping for the top (first) occurrence of silicon for a 1D cutline taken for an X value corresponding to the gate contact/region for loaded MASKVIEWS cutline data.

```
extract name="surface conc" surf.conc impurity="Net Doping" material="Silicon"
mat.occno=1 region="gate"
```

QUICKMOS 1D V_t

Extract the 1D threshold voltage of a p-type MOS cross section at $x=0.1$ using the built-in QUICKMOS 1D device simulator. This example uses a default gate bias setting of 0-5V for a 0.25V step with the substrate at 0V and a default device temperature of 300 Kelvin. Values of QSS and gate workfunction have also be specified.

```
extract name="1D Vt" ldvt ptype qss=1e10 workfunc=5.09 x.val=0.1
```

This 1D Vt extraction will calculate the 1D threshold voltage of an n-type MOS cross section at $x=0.1$, where a gate voltage range (0.5-20V) was specified while the substrate (Vb) is set at 0.2V. The device temperature has been set to 350 Kelvin.

```
extract name="1D Vt 0-20v" ldvt ntype bias=0.5 bias.step=0.25 bias.stop=20.0
vb=0.2 temp.val=350.0 x.val=0.1
```

Sheet Resistance and Sheet Conductance

Note: For sheet conductance extraction substitute "sheet.res" with "conduct" (e.g., conduct, p.conduct, n.conduct).

Extract the total sheet resistance of the first p-n region in the top (first) occurrence of polysilicon for a cutline at $x=0.1$. Polysilicon is treated as a metal by default but is flagged here as a semiconductor (semi.poly). The default device temperature of 300 Kelvin and no layer biases will be used and the incomplete ionization flag is also set for carrier freezeout calculations (see "Incomplete Ionization Of Impurities" physics section within the ATLAS manual).

```
extract name="Total SR" sheet.res material="Polysilicon" mat.occno=1 x.val=0.1
region.occno=1 semi.poly incomplete
```

Extract the n-type sheet resistance of the second p-n region in the top (first) occurrence of silicon for a cutline at $x=0.1$, where the second region is held at 4.0V and the device temperature is set to 325 Kelvin. These commands use the start/cont/done syntax to create a multi-line statement as described in Section 5.8: "Extract Features".

```
extract start material="Silicon" mat.occno=1 region.occno=2 bias=4.0 x.val=0.1
extract done name="N-type SR" n.sheet.res material="Silicon" mat.occno=1
temp.val=325 x.val=0.1 region.occno=2
```

The following multi-line statement extracts the p-type sheet resistance of the first p-n region in the top (first) occurrence of silicon for a cutline at $x=0.1$, where the first region is held at 5.0V. The second region is held at 1.0V and the first interface Qss value equal to $1e10$.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=5.0 x.val=0.1
extract cont material="Silicon" mat.occno=1 region.occno=2 bias=1.0 x.val=0.1
extract cont interface.occno=1 qss=1.0e10
extract done name="P-type SR" p.sheet.res material="Silicon" mat.occno=1
x.val=0.1 region.occno=1
```

Note: This is an example of the multi-line "start/continue/done" type of statement used to specify layer biases and Qss values. It is recommended that you always let the Extract popup write this particular syntax. The Qss value also specifies the material interface occurrence involved, counting from the top down. There can be any number of additional "continue" lines to specify the biases on other layers and the Qss values of other interfaces; the last line, "done", does the actual extraction.

1D Max/Min Concentration

Extract the peak concentration of net doping within the first p-n region of the top (first) layer of silicon for a 1D cutline at $x=0.1$.

```
extract name="Max 1d Net conc" max.conc impurity="Net Doping" material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1
```

Extract the peak concentration of phosphorus within any p-n regions (default) for all materials using a 1D cutline at $x=0.1$.

```
extract name="Max 1d phos conc" max.conc impurity="Phosphorus" material="All"
```

```
x.val=0.1
```

Extract the minimum concentration of boron within any p-n regions of the top (first) layer of silicon for a 1d cutline at x=0.1.

```
extract name="Min 1d bor conc" min.conc impurity="Boron" material="Silicon"
mat.occno=1 x.val=0.1
```

2D Max/Min Concentration

Extract the peak concentration of net doping for the entire 2D structure.

```
extract name="Max 2D net conc" 2d.max.conc impurity="Net Doping" material="All"
```

Extract the peak concentration of boron within the silicon material in the 2D “box” limits defined.

```
extract name="Max 2D bor conc" 2d.max.conc impurity="Boron" material="Silicon"
y.min=0.1 y.max=0.9 x.min=0.2 x.max=0.6
```

In addition to this statement, you can add the `interpolate` flag. When present, this flag causes the extraction to perform interpolation at the edges of the specified bounding box for min/max concentration and position.

Extract the minimum concentration of phosphorus for all materials within the 2D “box” limits. These limits are defined by user-defined *y* coordinates and *x* values corresponding to loaded MASKVIEWS cutline information for the specified electrode or region.

```
extract name="Min 2D phos conc" 2d.min.conc impurity="Phosphorus" material="All"
region="gate" y.min=0.1 y.max=0.9
```

The following multi-line extract command measures the minimum concentration of antimony for the entire 2D structure and return the x-y coordinates of the extracted concentration.

```
extract name="Min 2D ant conc" 2d.min.conc impurity="Antimony" material="All"
extract name="Min 2D ant conc X position" x.pos
extract name="Min 2D ant conc Y position" y.pos
```

Note: The x-y position syntax must directly follow the 2D concentration extraction (same as `start/continue/done` syntax). We advise you to use the Extract popup to create these statements.

2D Concentration File

The output file contains data of the format *x y c*, where *c* is the value of concentration at the coordinates *xy*. The following example extracts the boron concentration in Silicon for the whole structure.

```
extract 2d.conc.file material="silicon" impurity="boron" outfile="conc.dat"
```

1D Material Region Boundary

Extracting the maximum Y boundary (upper side) location of the first occurrence of silicon material for a 1d cutline taken at X=2.

```
extract name="max_y" max.bound material="silicon" x.val=2 mat.occno=1
```

Extracting the minimum X boundary (left side) location of the second occurrence of polysilicon material for a 1d cutline at Y=3.

```
extract name="min_x" min.bound material="polysilicon" y.val=3 mat.occno=2
```

2D Material Region Boundary

Extracting the minimum X boundary (left side) location of the photoresist material region at XY coordinates (7.6, -1.2).

```
extract name="minx" min.bound x.pos material="photoresist" x.val=7.6  
y.val=-1.2
```

Extracting the maximum Y boundary (upper side) location of the photoresist material region at XY coordinates (5.2, 0).

```
extract name="maxy" max.bound y.pos material="photoresist" x.val=5.2 y.val=0
```

2D Concentration Area

Integrates the Boron concentration within the specified “box” limits, using a cutline step of 0.05 microns.

```
extract name="limit_area" 2d.area impurity="Boron" x.step=0.05 x.min=0.01  
y.min=0.23 x.max=0.6 y.max=0.45
```

In addition to this statement, you can add the interpolate flag. When present, this flag causes the extraction to perform interpolation at the edges of the specified bounding box for min/max concentration and position.

Integrates the Phosphorus concentration for the whole 2D structure using a cutline step of 0.03 microns.

```
extract name="device_area" 2d.area impurity="Phosphorus" x.step=0.03
```

Note: The x.step refers to the number of 1d cutlines used to obtain the 2d area. For a device with an X axis of 7 microns, an x.step of 1 would result in 8 cutlines being used at 1 micron intervals.

2D Maximum Concentration File

Creates a Data format file plotting the position of the maximum potential, in silicon material only, for the whole 2D structure. A maximum potential Y position is found for every X step of 0.1 microns. These Data format files can be loaded into TONYPLOT (-ccd) to represent a line of maximum concentration across a device.

```
extract name="Total_max_pot" max.conc.file impurity="potential" x.step=0.1  
material="silicon" outfile="totalconc.dat"
```

Creates a Data format file plotting the position of the maximum potential, in any material, for the specified “box” limits. A maximum potential Y position is found for every X step of 0.2 microns.

```
extract name="limit_max_pot" max.conc.file impurity="potential"  
x.step=0.2 outfile="limitconc.dat" x.min=0 x.max=7 y.min=0 y.max=0.09
```

Note: The x.step does not refer to cutlines but to the number of X coordinates used. A value of 1 representing stepping 1 micron in X for every max Y value calculated.

QUICKMOS CV Curve

Extract a MOS CV curve, ramping the gate from 0 to 5 volts, with 0 volts on the backside and the device temperature set at 325 Kelvin (default 300 K). This example creates a curve that is stored in file cv.dat and can be shown using TONYPLOT. To bring up TONYPLOT on this file, an easy way is to highlight the file name and then click on DECKBUILD's **Tools** button. TONYPLOT starts and loads with the file automatically.

```
extract name="CV curve" curve(bias,ldcapacitance vg=0.0 vb=0.0 bias.ramp=vg
bias.step=0.25 bias.stop=5.0 x.val=0.1 temp.val=325) outfile="cv.dat"
```

To get the maximum capacitance for the same curve, insert the keyword `max` (by editing the syntax created by the popup). Notice that in this example, a single value is being extracted from a curve, not the curve itself. You still, however, store the curve used during the calculation into an output file, which is always the case.

```
extract name="CV curve Max cap" max(curve(bias,ldcapacitance vg=0.0 vb=0.0
bias.ramp=vg bias.step=0.25 bias.stop=5.0 x.val=0.1 temp.val=325))
outfile="cv.dat"
```

To find what the capacitance was at voltage 4.3 volts, use the following syntax:

```
extract name="MOS capacitance at Vg=4.3" y.val from curve(bias,ldcapacitance
vg=0.0 vb=0.0 bias.ramp=vg bias.step=0.25 bias.stop=5.0 x.val=0.1 temp.val=325)
where x.val = 4.3
```

The general form of this syntax is

```
extract y.val from curve(xaxis, yaxis) where x.val=number_on_xaxis
```

and:

```
extract x.val from curve(xaxis, yaxis) where y.val=number_on_yaxis
```

where `xaxis` and `yaxis` will determine the actual curve. The syntax for this example was created by using the popup to write the syntax for the CV curve, and then adding the `y.val... where x.val` syntax in the input deck.

For more examples on how to manipulate curves, see the examples in Section 5.4: “Device Extraction”.

Junction Capacitance Curve

Extract a curve of junction capacitance against bias where the first region in the top (first) layer of silicon is ramped from 0 to 5V. Capacitance of the first junction occurrence (upper) is measured and the resultant curve is output to the file `XjV.dat`. Device temperature is default (300 Kelvin). If only one junction exists for the selected region, you must use then a junction occurrence of one (upper).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=5.0 x.val=0.1 region.occno=1
extract done name="Junc cap vs bias" curve(bias,ldjunc.cap material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1 junc.occno=1) outfile="XjV.dat"
```

Extract the minimum junction capacitance on the created junction capacitance against bias curve. The second region in the top (first) layer of silicon is ramped from 0 to 3V and the capacitance of the second junction occurrence (lower) is measured. Device temperature is set for calculations to be 325 Kelvin. The resultant curve is output to the file `XjVmin.dat`, while the extracted minimum value is logged to the default results file (`results.final`).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=3.0 x.val=0.1 region.occno=2
extract done name="Junc cap vs bias" min(curve(bias,ldjunc.cap
material="Silicon" mat.occno=1 x.val=0.1 region.occno=2 junc.occno=2
temp.val=325)) outfile="XjVmin.dat"
```

Note: The junction occurrence is only valid for the specified region. In other words, there is only a maximum of two possible junctions for the specified region.

Junction Breakdown Curve

Extract a curve of electron ionization integral against bias where the first region in the top (first) layer of silicon is ramped from 0 to 5V and device temperature is set to be 325 Kelvin. The resultant breakdown curve is output to the file Nbreakdown.dat. See the “Impact” command section and “Impact Ionization” physics section in the ATLAS USER’S MANUAL for the Selberherr model used in calculation.

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=5.0 x.val=0.1 region.occno=1
extract done name="N Breakdown "curve(bias,n.ion material="Silicon" mat.occno=1
x.val=0.1 region.occno=1 temp.val=325)
outfile="Nbreakdown.dat"
```

The following extraction creates a curve of hole ionization integral against bias, and calculates the breakdown voltage corresponding to the point where the hole ionization integral intercepts 1.0. The second region in the top (first) layer of silicon is ramped from 0 to 20V and the device temperature is set to the default of 300 Kelvin. The resultant breakdown curve is output to the file Pbreakdown.dat and the breakdown voltage is appended to the default results file (results.final).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.50
bias.stop=20.0 x.val=0.1 region.occno=2
extract done name="P intercept" x.val from curve(bias,p.ion material="Silicon"
mat.occno=1 x.val=0.1 region.occno=2) where y.val=1.0
outfile="Pbreakdown.dat"
```

You can modify the selberherr model parameters using the syntax below. For more information, see Appendix A: “Models and Algorithms”.

```
extract start material="Silicon" mat.occno=1 bias=0.2 bias.step=0.08
bias.stop=5.0 x.val=0.3 region.occ=2
extract done name="iiP" curve(bias, p.ion material="Silicon" mat.occno=1
x.val=0.3 region.occno=2 egran=4.0e5 betap=1.0 betan=1.0
an1=7.03e5 an2=7.03e5 bn1=1.231e6 bn2=1.231e6 ap1=6.71e5 ap2=1.582e6
bp1=1.693e6 bp2=1.693e6) outfile="extract.dat"
```

SIMS Curve

Extract the concentration profile of net doping in the top (first) layer of silicon. The output curve is placed into the file SIMS.dat.

```
extract name="SIMS" curve(depth,impurity="Net Doping" material="Silicon"
mat.occno=1 x.val=0.1) outfile="SIMS.dat"
```

SRP Curve

Extract the **SRP** (Spreading Resistance Profile) in the top (first) silicon layer. The output curve is placed into the file SRP.dat.

```
extract name="SRP" curve(depth, srp materials="Silicon" mat.occno=1 x.val=0.1)
outfile="SRP.dat"
```

The following command will calculate the **SRP** (Spreading Resistance Profile) in the top (first) silicon layer using a specified 100 etch steps of uniform size. The output curve is placed into the file SRP100.dat.

```
extract name="SRP100" curve(depth,srp material="Silicon"
mat.occno=1 n.step=100 x.val=0.5) outfile="srp100.dat"
```

Note: Where `n.step` is not specified, the default is 50 etch steps of variable size dependent on the gradient of net concentration. If `n.steps` is set, uniform etch steps are used.

Sheet Resistance/Conductance Bias Curves

Extract the **Total** sheet conductance against bias curve of the first p-n region in the top (first) occurrence of polysilicon. Polysilicon is treated as a metal by default but is flagged here as a semiconductor (`semi.poly`). The device temperature is set to 325 Kelvin (default=300 Kelvin) and a bias ramped from 0 to 5V on the same polysilicon region.

```
extract start material="Polysilicon" mat.occno=1 bias=0.0 bias.step=0.00
bias.stop=5.0 x.val=0.1 region.occno=1
extract done name="Total SC" curve(bias,1dconduct material="Polysilicon"
mat.occno=1 temp.val=325 x.val=0.1 region.occno=1 semi.poly)
outfile="totalSC.dat"
```

Extract the n-type sheet conductance against bias curve of the first p/n region in the top (first) occurrence of silicon where a bias ramped from 0V to 5V on the same silicon region and a value of QSS (4.0e10) is specified for the first interface occurrence.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont interface.occno=1 qss=4.0e10
extract done name="N-type SC" curve(bias,1dn.conduct material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1) outfile="NtypeSC.dat"
```

Extract the p-type sheet conductance against bias curve of the first p-n region in the top (first) occurrence of silicon, where a bias ramped from 0 to 5V on the same silicon region and a bias of 2V is held on the first region of the top occurrence of polysilicon. A value of QSS (5.0e10) is also specified for the first interface occurrence.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont material="Polysilicon" mat.occno=1 bias=2.0 x.val=0.1
region.occno=1
extract cont interface.occno=1 qss=5.0e10
extract done name="P-type SC" curve(bias,1dp.conduct material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1) outfile="PtypeSC.dat"
```

The command below extracts the p-type sheet conductance against bias curve of the first and second p-n regions in the top (first) layer of silicon, where a bias is ramped from 1V to -2V on the top (first) polysilicon layer.

```
extract start material="Polysilicon" mat.occno=1 bias=1.0 bias.step=-0.05
bias.stop=-2.0 x.val=0.01
extract done name="region1+2" curve(bias,1dp.conduct material="Silicon"
mat.occno=1 x.val=0.01 region.occno=1 region.stop=2) outfile="region1+2.dat"
```

Note: For sheet resistance extraction, substitute "1dconduct" with "1dsheet.res" (i.e., 1dsheet.res, 1dnsheet.res, 1dpsheet.res).

Electrical Concentration Curve

Extract the electron distribution against depth for the top (first) layer of silicon where a bias is ramped from 0 to 5V for the first region of the silicon and a QSS of 4.0e10 set for the first interface occurrence. Device temperature is set at 325 Kelvin.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont interface.occno=1 qss=4.0e10
extract done name="Electrical conc" curve(depth,n.conc material="Silicon"
mat.occno=1 x.val=0.1 temp.val=325) outfile="extract.dat"
```

ED Tree (Optolith)

Create a Data format file plotting a single branch of an ED tree for deviation of 10% from the datum, the specified critical dimension (CD) value of 0.5. The `x.step` defines the defocus step to be used. 0.08 representing 8% of the total X axis range for each calculation. For each value of defocus at the specified critical dimension deviation, the value of dose is interpolated. Therefore, the resulting curve is dose against defocus for a critical dimension of 0.5 plus 10%.

```
extract name="ed+10" edcurve(da.value."DEFOCUS", da.value."CDs",
da.value."DOSE",dev=10 datum=0.5 x.step=0.08) outf="ed10.dat"
```

Note: If no `x.step` is specified the actual curve defocus points are used.

Elapsed time

The timer is reset to 10 seconds, a timestamp extracted before and then after a simulation. The elapsed time is then calculated by subtraction.

```
extract name="reset_clock" clock.time start.time = 10
extract name="t1" clock.time
<simulation>
extract name="t2" clock.time
extract name="elapsed_time" $t2 - $t1
```

Note: This extraction does not measure CPU time

5.4: Device Extraction

Device extraction always deals with a “logfile” that contains I-V information produced by a device simulator (such as ATLAS). Therefore, it deals almost exclusively in curves. The following section show how to construct a curve or extract values on a curve for all possible devices. For the special case of MOS devices, both ATLAS and SMINIMOS4 have popups with a number of pre-defined MOS tests. See Section 5.6: “MOS Device Tests” for more information.

Device extraction also deals with structure files, which contain information saved by a device simulator (e.g., ATLAS). You can extract this information by using the process extraction syntax style shown below. The following extracts the total electric field for silicon in a 1-D cutline, where $x = 0.5$ for the loaded device structure file.

```
extract name="test" 2d.max.conc impurity="E Field" material="Silicon" x.val=0.5
```

There are some differences between the syntax used by EXTRACT and the syntax used by the ATLAS output command. Section 5.10: “Using Extract with ATLAS” shows these differences.

EXTRACT allows you to construct a curve using separate X and Y axes. For each axis, you can choose the voltage or current on any electrode, the capacitance or conductance between any two electrodes, or the transient time for AC simulations. You can either manipulate the axes individually, such as multiplication or division by a constant, or combine axes in algebraic functions.

Note: The curve manipulation discussed is equally applicable to all curves, whether the curve came from process or device simulation. The only type-specific syntax relates to the curve axes. For example, gate voltage can't be extracted from a process simulator. If you try, then a warning message will appear.

5.4.1: The Curve

The basic element is always the curve. Once the curve is constructed, it can be used as is, by saving it to a file for use by TONYPLOT, or as an OPTIMIZER target, or it can be used as the basis for further extraction. For details on the extract curve syntax, see Section 5.3.1: “Extract Syntax”.

To construct a curve representing voltage on electrode “emitter1” (on the X axis) versus current on electrode “base2”, write:

```
extract name="iv" curve(v."emitter1", i."base2")
```

The first variable specified inside the parentheses becomes the X axis of the curve. The second variable becomes the Y axis. The v. “name” and i. “name” syntax is used for any electrode name — just insert the proper name of the electrode. The electrode name be defined previously (such as in the device deck, or previous to that in an ATHENA input deck using the electrode statement, or interactively in DEVEDIT). Electrode names may contain spaces but must always have quotation marks.

Transient time is represented by the keyword time.

```
extract name="It curve" curve(time, i."anode")
```

For Device temperature curves, use:

```
extract name="VdT" curve(v."drain", temperature)
```

For extracting a frequency curve use:

```
extract name="Idf" curve(i."drain", frequency)
```

To extract a capacitance or conductance curve, use this syntax:

```
extract name="cv" curve(c."electrode1"electrode2", v."electrode3")
```

and

```
extract name="gv" curve(g."electrode1"electrode2", v."electrode3")
```

For other electrical parameters (see Section 5.3.1: “Extract Syntax” section for valid electrical parameters) use the following syntax:

```
extract name="IdT" curve(elect."parameter",v."drain")
```

An `extract name` is given in each example. Although optional, it is always a good idea to name `extract` statements so they can be identified later. Names are always necessary for entering an `extract` statement in DECKBUILD’s OPTIMIZER, and for recognition by the VWF.

It is also possible to shift or manipulate curve axes. Each axis is manipulated separately. The simplest form of axis manipulation is algebra with a constant.

```
extract name="big iv" curve(v."gate"/50, 10*i."drain")
```

You can multiply, divide, add, or subtract any constant expression to each axis.

Curve axis can also be combined algebraically, similar to TONYPLOT’s function capability:

```
extract name="combine" curve(i."collector", i."collector"/i."base")
```

All electrode values (current, voltage, capacitance, conductance) can be combined in any form this way.

Another curve type is `deriv()` used to return the derivative (dy/dx). For example, statement below will create the curve of dy/dx gate bias and drain current plotted against and X axis of gate bias.

```
extract name="dydx" deriv(v."gate", i."drain")
outfile="dydx.dat"
```

It is also possible to calculate dy/dx to the n th derivative as below.

```
extract name="dydx2" deriv(v."gate", i."drain", 2)
outfile="dydx2.dat"
```

To find local maxima and minima on a curve, limit the section of the curve X axis. The following statement extracts the maximum drain current, where gate bias is between the limits of 0.5 volts and 2.5 volts.

```
extract name="limit" max(curve(v."gate", i."drain",x.min=0.5
x.max=2.5)) outf="limit.dat"
```

In addition, there are several operators which apply to curve axes. They are as follows:

```
abs(axis)
log(axis)
log10(axis)
sqrt(axis)
atan(axis)
-axis
```

For instance:

```
extract curve(abs(i."drain"), abs(v."gate"))
```

The operators can be combined. For example, `log10(abs(axis))`. These operators also work on curve axes from process simulation.

5.4.2: Curve Manipulation

A number of curve manipulation primitives exist:

```

min(curve)
max(curve)
ave(curve)
minslope(curve)
maxslope(curve)
slope(line)
xintercept(line)
yintercept(line)
area from curve
area from curve where x.min=X1 and x.max=X2
x.val from curve where y.val=k
y.val from curve where x.val=k
x.val from curve where y.val=k and val.occno=n
y.val from curve where x.val=k and val.occno=n
grad from curve where y.val=k
grad from curve where x.val=k

```

For details on Extract curve manipulation syntax, see Section 5.3.1: “Extract Syntax”.

For instance, using the BJT curve example above, you could find the maximum of I_c/I_b vs I_c , or maximum beta, by writing:

```
extract name="max beta" max(curve(i."collector", i."collector"/i."base"))
```

`max()`, `min()`, and `ave()` all work on the Y axis of the curve.

The sloped lines and intercepts often work together. The primitives `minslope()` and `maxslope()` can be thought of as returning a line. Extracting a line by itself has no meaning, so three other operators take a line as input. The operators are `slope()`, which returns the slope of the line, and `xintercept()` and `yintercept()`, which return the value where the line intercepts the corresponding axis.

For instance, a V_t test for MOS devices looks at a curve of V_g (x) versus I_d (y) and finds the X intercept of the maximum slope. Such a test would look like:

```
extract name="vt" xintercept(maxslope(curve(abs(v."gate", abs(i."drain"))))
```

Some V_t tests take off $V_d/2$ from the resulting value. You could write:

```
extract name="vt" xintercept(maxslope(curve(abs(v."gate",
abs(i."drain")))) - ave(v."drain")/2
```

Note that the last example uses:

```
ave(v."drain")/2
```

The `max()`, `min()`, and `ave()` operators can be used on both curves,

```
extract name="Iave" ave(curve(v."gate", i."drain"))
```

and also on individual curve axes,

```
extract name="Iave" ave(i."drain")
```

or even on axis functions:

```
extract name="Icb max" max(i."collector"/i."base")
```

You can also find the Y value on a curve for a given X value and the other way round. For example, to find the collector current (Y) for base voltage 2.3 (X), use:

```
extract name="Ic[Vb=2.3]" y.val from curve(abs(v."base"), abs(i."collector"))
where x.val = 2.3
```

EXTRACT uses linear interpolation if necessary. If more than one point on the curve matches the condition, EXTRACT takes the first one, unless you use the following syntax to specify the occurrence of the condition. This example would find the second Y point on the curve matching an X value of 2.3.

```
extract name="Ic[Vb=2.3]" y.val from curve(abs(v."base",
abs(i."collector")))
where x.val = 2.3 and val.occno =2
```

The condition used for finding an intercept can be a value or an expression and therefore use the min(), max(), and ave() operators. The following command creates a transient time against drain-gate capacitance curve and calculates the intercepting time where the capacitance is at its minimum value.

```
extract name="t at Cdrain-gate[Min]" x.val from curve(time, c."drain"gate")
where y.val=min(c."drain"gate")
```

In addition to finding intercept points on curves, you can also calculate the gradient at the intercept, specified by either a Y or X value as shown below.

```
extract name="slope_at_x" grad from curve(v."gate", i."drain")
where x.val=1.5
extract name="slope_at_y" grad from curve(v."gate", i."drain")
where y.val=0.001
```

You can also find the area under a specified curve for either the whole curve or as below between X limits.

```
extract name="iv area" area from curve(v."gate", c."drain"gate")
where x.min=2 and x.max=5
```

5.4.3: BJT Example

As a final example for device extraction, consider finding, say, the beta value for a BJT device, at 1/10th the current for max beta. This example sums up the information presented so far, and also introduces the feature of variable substitution.

First, you need to figure out what the current is at max beta. Max beta was presented in a previous example:

```
extract name="max beta" max(curve(i."collector", i."collector"/i."base"))
```

After this statement has been run, extract remembers the extract name, max beta, and the resulting value. Use this information later on using variable substitution. In this example, you need to get the current, or X axis value, at max beta, to figure out what 1/10th of it is. To do this, use the extracted max beta as our Y axis “target value”:

```
extract name="Ic[max beta]" x.val from curve(i."collector",
i."collector"/i."base") where y.val=$"max beta"
```

Finally, extract the value of Ic/Ib for Ic=max beta/10.

```
extract name="Ic/Ib for Ic=Ic[max beta]/10" y.val from curve(i."collector",
i."collector"/i."base") where x.val=$"Ic[max beta]/10"
```

For more information about variable substitution, see Section 5.8: “Extract Features”.

5.5: General Curve Examples

The following examples assume that they are extracting values from the currently loaded logfile running under DECKBUILD. Saved “IV” log files, however, can be used directly with `extract` using the syntax below.

```
extract init infile="filename"
```

Note: You can enter `extract` commands on multiple lines using a backslash character for continuation. You should, however, enter the syntax shown below on a single line although shown on two or more lines.

5.5.1: Curve Creation

The following command extracts a curve of collector current against base voltage and places the output in `icvb.dat`.

```
extract name="IcVb curve" curve(i."collector", v."base") outfile="icvb.dat"
```

5.5.2: Min Operator with Curves

The following command calculates the minimum value for a curve of drain current against internal gate voltage.

```
extract name="Vgint[Min]" min(curve(i."drain", vint."gate"))
```

5.5.3: Max Operator with Curves

The following command calculates the maximum value for a curve of base voltage against base-collector capacitance.

```
extract name="Cbase-coll[Max]" max(curve(v."base", c."base"collector))
```

5.5.4: Ave Operator with Curves

The following command calculates the average value for a curve of drain current against gate-drain conductance.

```
extract name="Ggate-drain[Ave]" ave(curve(i."drain", g."gate"drain))
```

5.5.5: X Value Intercept for Specified Y

The following command creates a frequency against drain current curve and calculates the intercepting frequency for a drain current of 1.5×10^{-6} .

```
extract name="Freq at Id=1.5e-6" x.val from curve(frequency, i."drain") where  
y.val=1.5e-6
```

5.5.6: Y Value Intercept for Specified X

The following command creates a drain voltage against device temperature curve and calculates the intercepting temperature for a drain voltage of 5V.

```
extract name="T at Vd=5" y.val from curve(v."drain", temperature) where  
x.val=5.0
```

5.5.7: Abs Operator with Axis

The following command creates a curve of absolute gate voltage against absolute optical wavelength (log, log10 and sqrt also available).

```
extract name="Vg-optW curve" curve(abs(v."gate"), abs(elect."optical wavelength"))
```

5.5.8: Min Operator with Axis Intercept

The following command creates a transient time against gate-drain capacitance curve and calculates the intercepting time where the capacitance is at its minimum value.

```
extract name="t at Cgate-drain[Min]" x.val from curve(time, c."gate" "drain")
where y.val=min(c."gate" "drain")
```

5.5.9: Max Operator with Axis Intercept

The following command creates a collector current against collector current divided by base current curve and calculates the intercepting collector current where I_c/I_b is at a maximum value.

```
extract name="Ic at Ic/Ib[Max]" x.val from curve(i."collector", i."collector"/
i."base") where y.val=max(i."collector"/i."base")
```

5.5.10: Second Intercept Occurrence

The following command creates a gate voltage against source photo current curve and calculates the second intercept of gate voltage for a source photo current of $2e-4$.

```
extract name="2nd Vg at Isp=2e-4" x.val from curve(v."gate", elect."source photo current")
where y.val=2e-4 and val.occno=2
```

5.5.11: Gradient at Axis Intercept

The following command creates a probe Itime against drain current curve and finds the gradient at the point where probe Itime is at a maximum.

```
extract name="grad_at_maxTime" grad from curve(probe."Itime",
i."drain") where y.val=max(probe."Itime")
```

5.5.12: Axis Manipulation with Constants

The following command creates a gate voltage divided by ten against total gate capacitance multiplied by five. Adding and subtracting are also available.

```
extract name="Vg/10 5*C-gg curve" curve(v."gate"/10, 5*c."gate" "gate")
```

5.5.13: X Axis Interception of Line Created by Maxslope Operator

The following command calculates the X axis intercept for the maximum slope of a drain current against gate voltage curve.

```
extract name="Xint for IdVg" xintercept(maxslope(curve(i."drain", v."gate")))
```

5.5.14: Y Axis Interception of Line Created by Minslope Operator

The following command calculates the Y axis intercept for the minimum slope of a substrate current against drain voltage.

```
extract name="Yint for IsVd" yintercept(minslope(curve(i."substrate",
v."drain")))
```

5.5.15: Axis Manipulation Combined with Max and Abs Operators

The following command calculates the maximum value of drain-gate resistance.

```
extract name="Rdrain-gate[Max]" max(1.0/(abs(g."drain" "gate")))
```

5.5.16: Axis Manipulation Combined with Y Value Intercept

The following command creates a gate voltage against drain-gate resistance and calculates the intercepting drain-gate resistance for a gate voltage of 0V.

```
extract name="Rdrain-gate at Vg=0" y.val from curve (v."gate", 1.0/
abs(g."drain" "gate"))
where x.val=0.0
```

5.5.17: Derivative

The following command creates the curve of dydx gate bias and drain current plotted against and X axis of gate bias.

```
extract name="dydx" deriv(v."gate", i."drain")
outfile="dydx.dat"
```

This further example calculates to the 2nd derivative.

```
extract name="dydx2" deriv(v."gate", i."drain", 2)
outfile="dydx2.dat"
```

5.5.18: Data Format File Extract with X Limits

The following command finds the local maximum in Data Format file for the curve of vin between 2 and 5 volts against power.

```
extract name="max[2-5]" max(curve(da.value."vin", da.value."power",
x.min=2 x.max=5)) outf="max2-5.dat"
```

5.5.19: Impurity Transform against Depth

The following command calculates the electron concentration in the first occurrence of silicon material for a cutline of X=1 squared against depth.

```
{fixed} extract name="nconc^2" curve(depth,(n.conc material="Silicon"
mat.occno=1 x.val=1) * (n.conc material="Silicon" mat.occno=1 x.val=1))
outfile="nconc.dat"
```

5.6: MOS Device Tests

A list of ready-made MOS extract statements is also provided. Use them directly or make modifications to suit testing needs. DECKBUILD allows you to create, modify, and save tests.

The following MOS tests are:

- Vt
- Beta
- Theta
- Leakage
- Bvds
- Idsmax
- SubVt
- Isubmax
- Vg[Isubmax]

Do the following to access the list of MOS extract routines.

- **ATLAS:** Choose **Commands**→**Extracts**→**Device...** and the ATLAS Extraction popup will appear. Choose the desired test and click on the **WRITE** button to insert the test into the input deck. Using the **User defined** option, you can enter custom extracts into the popup and save them as defaults.

When you click the **Write Deck** button on the **Control** popup, the extract syntax will be written automatically to the deck along with the selected tests (Figure 5-5).

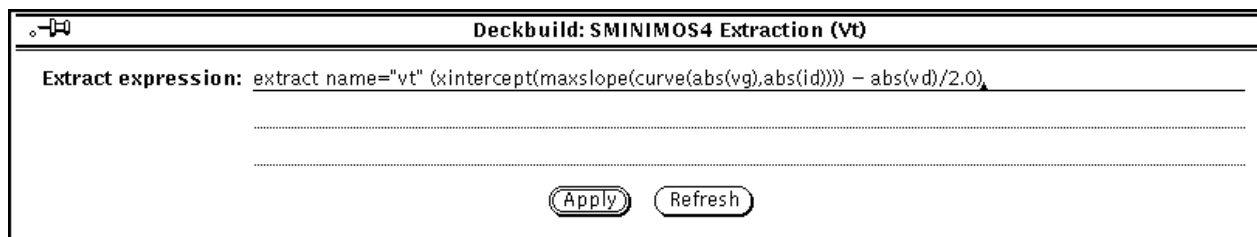


Figure 5-5: The ATLAS Extraction (Vt) Popup

5.7: Extracted Results

Extracted results appear both with the simulator output in the **tty** subwindow and in a special file named by default `results.final`. You can name the file using the `datafile="filename"` syntax. Use the file to compare the results from a large number of runs. For example, if using DECKBUILD's built-in optimizer, the file gives a concise listing of all the results as a function of the input parameters. The extract results file is created in the current working directory.

5.7.1: Units

- Material thickness (angstroms)
- Junction depth (microns)
- Impurity concentrations (impurity units, typically atoms/cm³)
- Junction capacitance (Farads/cm²)
- QUICKMOS capacitance (Farads/cm²)
- QUICKMOS 1D V_t (Volts)
- QUICKBIP 1D solver (see the QUICKBIP section)
- Sheet resistance (Ohm/square)
- Sheet conductance (square/Ohm)
- Electrode voltage (Volts)
- Electrode internal voltage (Volts)
- Electrode current (Amps)
- Capacitance (Farads/micron)
- Conductance (1/Ohms)
- Transient time (Seconds)
- Frequency (Hertz)
- Temperature (Kelvin)
- Luminescent power (Watts/micron)
- Luminescent wavelength (Microns)
- Available photo current (Amps/micron)
- Source photo current (Amps/micron)
- Optical wavelength (Microns)
- Optical source frequency (Hertz)
- Current gain (dB)
- Unilateral power gain frequency (dB)
- Max transducer power gain (dB)

If desired, you can perform whatever unit shifting required by adding the appropriate constants in the device extract tests and saving them as the default. The units are always printed out along with the extract results for built-in single value extract routines. Custom extract routines do not show units.

5.8: Extract Features

5.8.1: Extract Name

extract statements should almost always be given names. The name must be prepended to the remainder of the extract statement. For example:

```
extract name="gateox thickness" oxide thickness x.val=1.0
```

The extract name is used in three ways. The name appears on the OPTIMIZER worksheet when you enter the extract statement as a target, and on the VWF worksheet as an extracted parameter. It can also be used in further extract statements to perform variable substitution. The name can contain spaces.

5.8.2: Variable Substitution

The extract parser maintains a list of variables, each of which consists of a name and a value. A name is defined by any previous named extract statement. The corresponding value is the result of the statement.

To refer to a variable's value, precede it with a '\$'. Quotes are optional around variable references, except when the variable name contains spaces, in which case the \$ must precede the quotes. The substituted variable acts as a floating point number, and can be used in any extract expression that uses numerical arguments.

For example:

```
extract name="xj1" xj silicon junc.occno=1
extract name="xj2" xj silicon junc.occno=2
extract name="deltaXj" abs($xj1 - $xj2)
```

Examples with spaces:

```
extract name="max boron" max.conc boron
extract name="max arsenic" max.conc arsenic
extract name="PN ratio" $"max boron"/$"max arsenic"
```

You can also use variable substitution in extract with the set command as shown below.

```
set cutline=0.5
extract name="gateox thickness" oxide thickness x.val=$cutline
```

In addition, filenames to be loaded can also be specified this way. For example:

```
set efile = structure.str
extract init infile="$'efile'"
```

Note: Single quotes can be used to substitute where \$-variable must appear within double quotes.

5.8.3: Min and Max Cutoff Values

Statements may contain min.val=value or max.val=value or both to define a valid range for extracted results (single-valued results only, not curves). If you do not define either max or min, then the range extends from +infinity to the stated value respectively. If the extracted value is outside the range, then an error message is printed along with the extracted results and also appended to the default results file.

5.8.4: Multi-Line Extract Statements

EXTRACT statements may be spread over multiple lines to specify layer biases and QSS values as shown in above examples. This involves using the `start/cont/done` syntax.

5.8.5: Extraction and the Database (VWF)

When run with the VIRTUAL WAFER FAB, all extract values in the deck appear as output result columns on the split worksheet. Each row of the worksheet contains the input parameters used to create the results. The extracted value cell values are filled in automatically as the split points complete. If some extracts are only intermediate calculations and are not required to be included in the results worksheet the hide flag can be used. This prevents unrequired extract results from cluttering the worksheet data.

The min/max extract ranges, if defined, are examined. If any extracted value is out of range, then children of that deck fragment (any part of the worksheet that uses the simulation results of that deck fragment) are automatically de-queued and marked with a parent error. The fragment is marked with a range error. The purpose here is that the system does not waste its time by running any simulation beyond that point in the input deck where the range error occurred, for all parts of the split tree that use the particular values of the deck.

5.9: QUICKBIP Bipolar Extract

QUICKBIP is a 1D simulator for bipolar junction transistors (BJT) and is fully integrated inside the DECKBUILD environment. It is accessed using the `extract` command and is available for use with any SILVACO simulator.

The doping profile passed to the QUICKBIP solver should be a bipolar profile. At least, three regions must exist. The top region in the first silicon layer is taken to be the emitter. There may be other materials on top of the silicon.

QUICKBIP can be used with either ATHENA (2-D process simulation) or SSUPREM3 (1-D process simulation). It is used in cases where a 1-D device simulation is both easier and faster to turn around a result. Examples of the QUICKBIP `extract` command language are listed as follows:

```
extract name="bip test bf" bf
extract name="bip test nf" nf
extract name="bip test is" gpis
extract name="bip test ne" ne
extract name="bip test ise" ise
extract name="bip test cje" cje
extract name="bip test vje" vje
extract name="bip test mje" mje
extract name="bip test rb" rb
extract name="bip test rbm" rbm
extract name="bip test irb" irb
extract name="bip test tf" tf
extract name="bip test cjc" cjc
extract name="bip test vjc" vjc
extract name="bip test mjc" mjc
extract name="bip test ikf" ikf
extract name="bip test ikr" ikr
extract name="bip test nr" nr
extract name="bip test br" br
extract name="bip test isc" isc
extract name="bip test nc" nc
extract name="bip test tr" tr
```

Any name can be assigned to each command. In the case of a 2-D simulator, the lateral position of the vertical profile has to be specified with the parameter `x.val=n`. For example:

```
extract name = "forward transit time" tf x.val=0.3
```

Alternatively, a boolean region can be specified when running in conjunction with the IC Layout interface. For example:

```
extract name="my test" tf region="pnp_active_poly"
```

In this case, the bipolar test is performed only in the case where an IC layout cross section intersects the named region.

You can modify QUICKBIP tuning parameters for using the syntax shown below. Appendix A: “Models and Algorithms” provides a more detailed explanation.

```
extract name="Tuning bf" bf x.val=0.5 bip.tn0=1.0e-5 bip.tp0=1.0e-3
bip.an0=2.9e-31 bip.ap0=0.98e-31 bip.nsrhn=5.0e12 bip.nsrhp=5.0e15
bip.betan=2.1 bip.betap=1.
```

Table 5-1 shows the `extract` parameters representing the BJT parameters.

Table 5-1: BJT Parameters

Parameter	Description	Units
bf	Ideal Maximum Forward Beta	
nf	Forward current Emission Coefficient	
gpis	Transport saturation current (IS)	A/cm2
ne	Base-Emitter Leakage Emission Coefficient	
ise	Base-Emitter Leakage Saturation Current	A/cm2
cje	Base-Emitter Zero Bias DEpletion Capacitance	F/cm2
vje	Base-Emitter built in potential	V
mje	Base-Emitter exponential factor	
rb	Zero bias base resistance	Ohms/square
rbm	Minimum base resistance at high current	Ohms/square
irb	Current at half base resistance value	A/cm2
tf	Ideal forward transit time (1/ft)	secs
cjc	Base-Collect zero bias depletion capacitance	F/cm2
vjc	Base-Collector built in potential	V
mjc	Base-Collector exponential factor	
ikf	Corner of Forward Beta High current roll-off	A/cm2
ikr	Corner of Reverse Beta High current roll-off	A/cm2
nr	Reverse Current Emission Coefficient	
br	Ideal Maximum Reverse Beta	
isc	Base-Collector Leakage Saturation Current	A/cm2
nc	Base-Emitter Leakage Emission Coefficient	
tr	Ideal forward transit time	secs

Automated command writing is accomplished with the use of the **DeckBuild Extract** popup window. This is accessed from the **Commands** menu when either SSUPREM3 or ATHENA is selected as the current simulator.

I-V Curves can be visualized with TONYPLOT if the **Compute I-V curve** option is selected on the EXTRACT popup. In this case, select from either forward or reverse characteristics and specify the axes of the curve.

- All extracted parameters can be used as optimization targets.
- All extracted parameters are appended to the default results file in the current working directory. Unless specified, using the `datafile=filename` syntax, it defaults to `results.final`.
- When running under the VWF, all extracted parameters will be logged for regression modeling.

QUICKBIP solves fundamental system of semiconductor equations, continuity equations for electrons and holes, and Poisson's equation for potential self-consistently using the Gummel method. The following physical models are taken into account by QUICKBIP:

- Doping-dependent mobility
- Electric field dependent mobility
- Band gap narrowing
- Shockley-Read-Hall recombination
- Auger recombination

QUICKBIP is fully automatic so that it is unnecessary to specify input biases. QUICKBIP calculates both forward and inverse characteristics of the BJT. For an n-p-n device, these sets are as follows:

1. $V_{eb} = -0.3 \dots -V_{eb_final}$, $V_{eb_step} = -0.025$, $V_{cb} = 0$ V
2. $V_{cb} = -0.3 \dots -V_{cb_final}$, $V_{cb_step} = -0.025$, $V_{eb} = 0$ V
3. V_{eb_final} and V_{cb_final} depend on the particular BJT structures, usually about $-1 \dots -1.5$ (high injection level).

For a p-n-p device, all signs are changed.

5.10: Using Extract with ATLAS

Do the following to calculate extract parameters during an ATLAS simulation.

1. Include an output statement in your original input deck that specifies the parameters of interest (e.g., output charge to specify charge concentration). You cannot extract a parameter unless you specify that parameter (either explicitly or by default) in an output statement.
2. Insert an extract statement to extract the desired parameters (see Chapter 5: “Extract”).

There are some differences between the EXTRACT syntax and the syntax used by the ATLAS output statement. To extract parameters, use the correct extract statement syntax (not the ATLAS Output statement syntax). For example, the ATLAS Output statement uses E.Field to specify electric field, while the extract statement requires the name E.Field. To extract electric field include the following lines in the input deck:

```
Output E.Field
...
...
...
Extract ... Impurity="E Field"
```

The following table shows the differences between the ATLAS syntax and the extract statement syntax.

ATLAS Parameter	ATLAS Default	Extract Parameter	Units
POTENTIAL	Mandatory	Potential	V
NET DOPING	Mandatory	Net Doping	atoms/cm ³
ELECTRON CONCENTRATION	Mandatory	Electron Conc	cm ³
HOLE CONCENTRATION	Mandatory	Hole Conc	cm ³
CHARGE	False	Change Conc	atoms/cm ³
CON.BAND	False	Conduction Band Energy	V
E.FIELD/EFIELD	False	E Field	V/cm
E.MOBILITY	False	e- Mobility	cm ² /Vs
E.TEMP	True	Electron Temp	K
E.VELOCITY	False	Electron Velocity	cm/s
EX.FIELD	True	E Field X	V/cm
EX.VELOCITY	False	e- Velocity X	m/s
EY.FIELD	False	E Field Y	V/cm
EY.VELOCITY	False	e- Velocity Y	m/s
FLOWLINES	False	Current Flow	None
H.MOBILITY	False	h+ Mobility	cm ² /Vs
H.TEMP	True	Hole Temp	K
H.VELOCITY	False	Hole Velocity	cm/s
HX.VELOCITY	False	h+ Velocity X	m/s
HY.VELOCITY	False	h+ Velocity Y	m/s

ATLAS Parameter	ATLAS Default	Extract Parameter	Units
IMPACT	True	Impact Gen Rate	scm ³
JY.ELECTRON	False	Je- Y	A/cm ²
J.ELECTRON	True	Je- Current Magnitude	A/cm ²
JX.ELECTRON	FALSE	Je- X	A/cm ²
J.CONDUC	True	Conduction Current	A/cm ²
J.DISP	False	Displacement Current	A/cm ²
J.HOLE	True	h+ Current Magnitude	A/cm ²
J. TOTAL	True	Total Current Density	A/cm ²
JX.CONDUC	False	Cond Current X	A/cm ²
JX.HOLE	False	Jh+ X	A/cm ²
JX. TOTAL	False	Jtot X	A/cm ²
JY.CONDUC	False	Cond Current Y	A/cm ²
JY.HOLE	False	Jh+ Y	A/cm ²
JY. TOTAL	False	Jtot Y	A/cm ²
PHOTOGEN	True	Photo Generation Rate	scm ³
QFN	True	Electron QFL	V
QFP	True	Hole QFL	V
QSS	False	Interface Charge	cm ²
RECOMB	True	Recombination Rate	scm ³
TOT.DOPING	False	Total Doping	atoms/cm ³
TRAPS	True	Traps	cm ³
U.AUGER	False	Auger Recomb Rate	scm ³
R.RADIATIVE	False	Radiative Recomb Rate	scm ³
U.SRH	False	SRH Recomb Rate	scm ³
VAL.BAND	False	Valence Band Energy	V
X.COMB	False	Composition X	None
Y.COMB	True	Composition Y	None
OPT.INTENS	False	Optical Intensity	W/cm ²
OX.CHARGE	False	Fixed Oxide Charge	cm ³

6.1: Overview

The OPTIMIZER is a mechanism that automatically varies one or more input parameters to obtain simulated results, using those parameters which match one or more targets. The OPTIMIZER runs through a number of iterations until the results match the targets within a certain tolerance.

The OPTIMIZER uses the Modified Levenberg-Marquart algorithm to build a response surface of results versus input parameters as the iterations progress. This response surface is used to calculate the input parameter values for each iteration. If for some reason the OPTIMIZER cannot achieve convergence, it stops and display the error condition.

In practice, several parameters are measured and the simulation is then tuned to those values. For example, MOS structure, gate oxide thickness, Vt curves, I-V curves, and a SIMS profile can all be used to tune the input deck with the OPTIMIZER.

The OPTIMIZER is controlled through an easy-to-use graphical worksheet. The worksheet allows you to enter and edit input parameters, targets, and setup information (such as error tolerance) used by the OPTIMIZER. There is also a real-time graphical results display that visualizes input parameter values, target values, and error terms so you can easily track the OPTIMIZER's iterations.

6.1.1: Features

The OPTIMIZER eliminates guesswork by determining the input parameter values necessary to match one or more targets quickly and accurately. Furthermore, since the OPTIMIZER is built on top of DECKBUILD's native auto-interfacing capability, parameters in one simulator can be optimized against the extracted results from a different simulator. For example, optimizing can be against a Vt measurement and an I-V curve simulated in ATLAS using process input parameters in SSUPREM3 or ATHENAor both.

The OPTIMIZER is most useful for tuning studies. Use it to calibrate extracted simulation results to measured data by changing coefficients, such as diffusion and segregation; rather than trying to vary settings, such as simulated time and temperature. You can use such well-tuned input deck fragments, each of which performs a known operation, to build entire input decks. For design studies, rather than tuning studies, we recommend the VIRTUAL WAFER FAB (VWF). The VWF constructs and allows visualization of the entire process response surface, opposed to meeting a single target. It also acts in unison with the OPTIMIZER by storing the well-tuned input deck operations for later use.

Since the input deck requires no modification and no special statements, you can easily optimize any existing deck. When there are satisfying optimization results, the OPTIMIZER can copy the final parameter values back into the deck.

You can save all parameter and target data from the worksheet to disk and reload it at any time.

6.1.2: Terminology

This chapter makes reference to input parameters and targets. Input parameters, or just parameters, are any numerical constants in the input deck. Examples include implant energy, diffusion time, and gate voltage. Targets are values (either single points or entire curves) that are extracted from the simulated results. Examples include oxide thickness, Vt, and a curve of net concentration versus depth.

6.2: Using The Optimizer

6.2.1: Overview

Using the OPTIMIZER is easy. You must perform the following steps once an input deck runs with DECKBUILD.

1. Define the input parameters
2. Define the targets
3. Optionally, define the setup information
4. Start the optimization

6.2.2: The Optimizer Window

The Optimizer is controlled from the **DeckBuild Optimizer** window (Figure 6-1). To display the **Optimizer** window, select **Main Control**→**Optimizer...** in DECKBUILD.

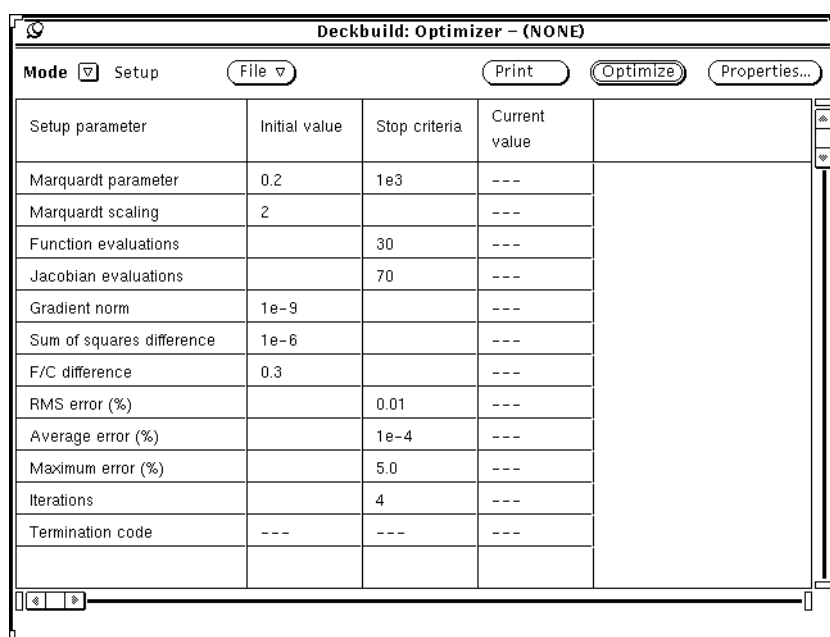


Figure 6-1: Optimizer Window

6.2.3: Optimization Modes

The **Mode** menu is located at the top left of the **Optimizer** window (Figure 6-2). This menu is used to select the five screens displayed in the **Optimizer** window.

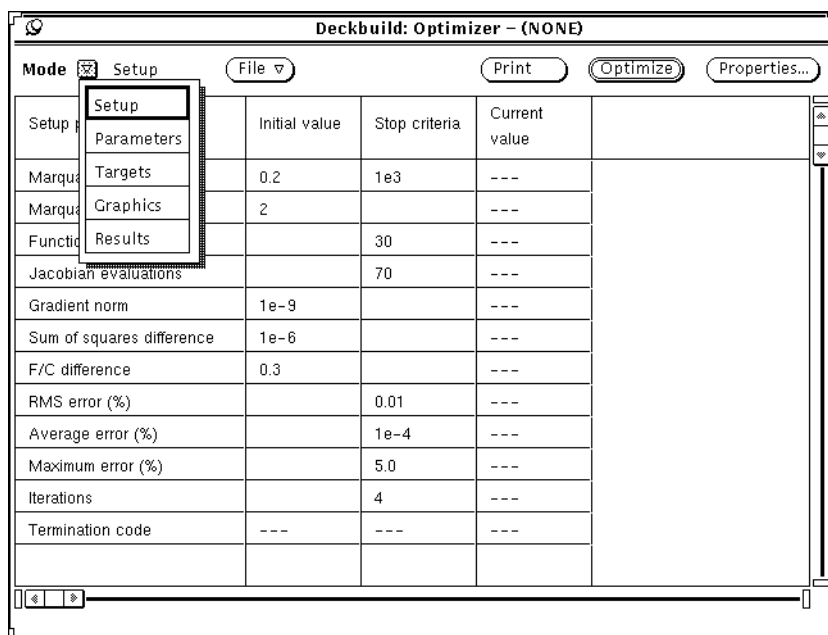


Figure 6-2: Optimizer Modes

The five modes in this window are:

- **Setup** — Optimization setup worksheet.
- **Parameters** — Input parameter definition worksheet.
- **Targets** — Target definition worksheet.
- **Graphics** — Graphical visualization of the optimization run.
- **Results** — Worksheet of parameter and target values for each iteration of a run.

You can define parameters, targets, and setup information for an optimization run in any order. The OPTIMIZER, however, does not run until you define at least one parameter and one target.

6.2.4: Optimizing

Once the parameter, target, and optional setup information is defined, the OPTIMIZER can be started by clicking on the **Optimize** button. Clicking on the button a second time aborts the optimization.

Once the optimization has started, all worksheet information, as well as the input deck, is set to a read-only state that cannot be edited.

During the optimization run, the **Graphics** mode shows real-time updates of current parameter, target, and error values.

At the beginning of a run, the OPTIMIZER always runs a sensitivity analysis of $n+1$ loops at the start of the input deck. n is the number of input parameters.

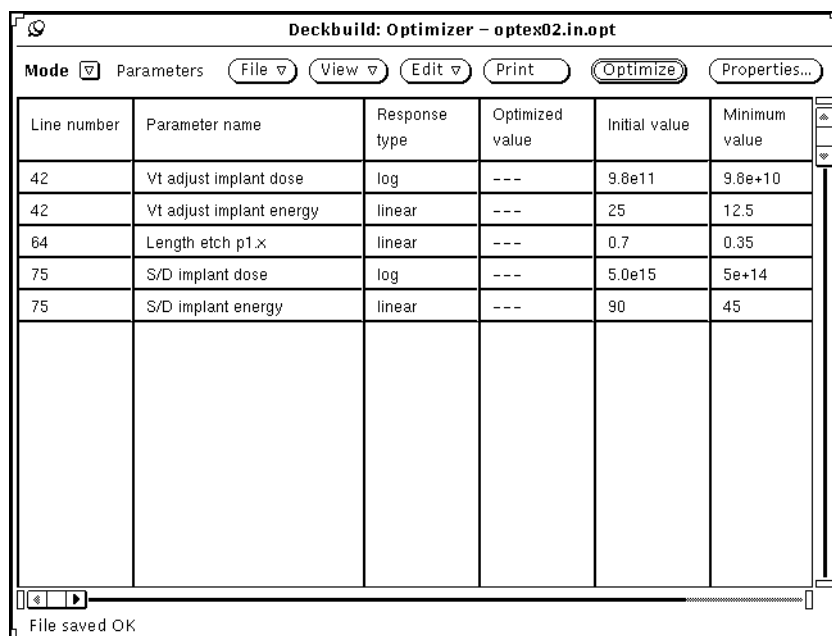
6.3: Parameters

A parameter is any numeric constant in an input deck. That constant may represent anything, such as etch thickness, contact voltage, or work function. Because the OPTIMIZER needs to vary its value to achieve convergence, the parameter must be numeric.

6.3.1: Adding Parameters

To add a parameter, follow these four steps:

1. Display the Parameter worksheet by setting **Mode to Parameters**. To do this, position the pointer over **Mode**, click the MENU mouse button, and select **Parameters**. The **Parameter** worksheet is then displayed (Figure 6-3).



Line number	Parameter name	Response type	Optimized value	Initial value	Minimum value
42	Vt adjust implant dose	log	---	9.8e11	9.8e+10
42	Vt adjust implant energy	linear	---	25	12.5
64	Length etch p1.x	linear	---	0.7	0.35
75	S/D implant dose	log	---	5.0e15	5e+14
75	S/D implant energy	linear	---	90	45

Figure 6-3: Parameter Worksheet with 5 Parameters Defined

2. Select (highlight) the line in the input deck containing the parameter(s) to be added. To do this, position the pointer over the line, and triple-click the SELECT mouse button to capture the entire line. The OPTIMIZER also accepts lines that have only a word or some selected characters.
3. Choose **Add** from the **Edit** pull-down menu. To do this, move the pointer over the **Edit** button, click on MENU and select **Add** (Figure 6-4).

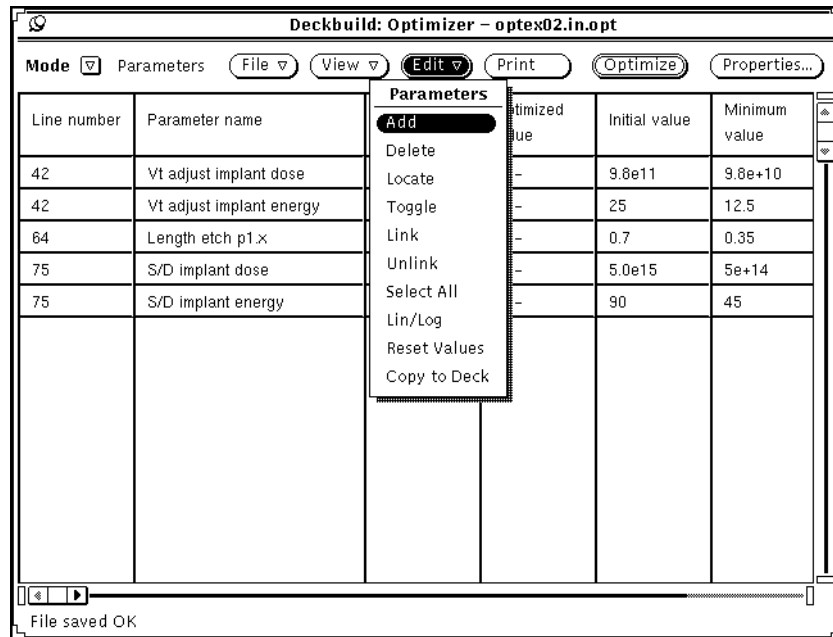


Figure 6-4: Parameter Edit Menu

After selecting **Add**, the **Parameter define** popup appears (Figure 6-5).

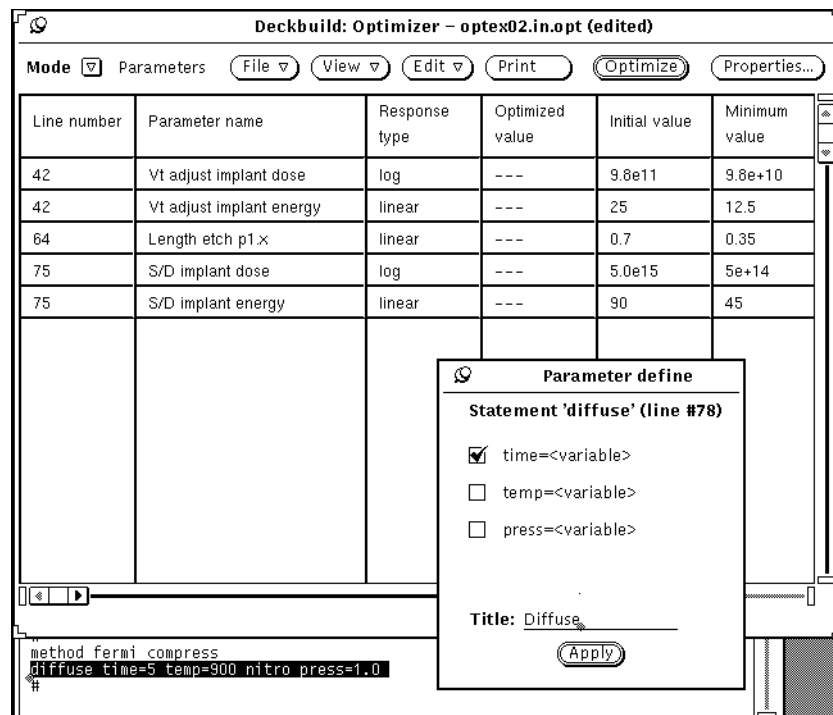


Figure 6-5: Parameter Define Popup

4. Check the checkboxes next to the desired parameter(s) by clicking SELECT next to their names. You can also type in a title. The parameter name on the worksheet are formed by appending the individual parameter names (shown next to the checkboxes) with the title. When you select all the desired parameters, click on **Apply**. A new row is inserted into the **Parameter** worksheet for each selected parameter on the popup (Figure 6-6). The rows are always arranged by line number.

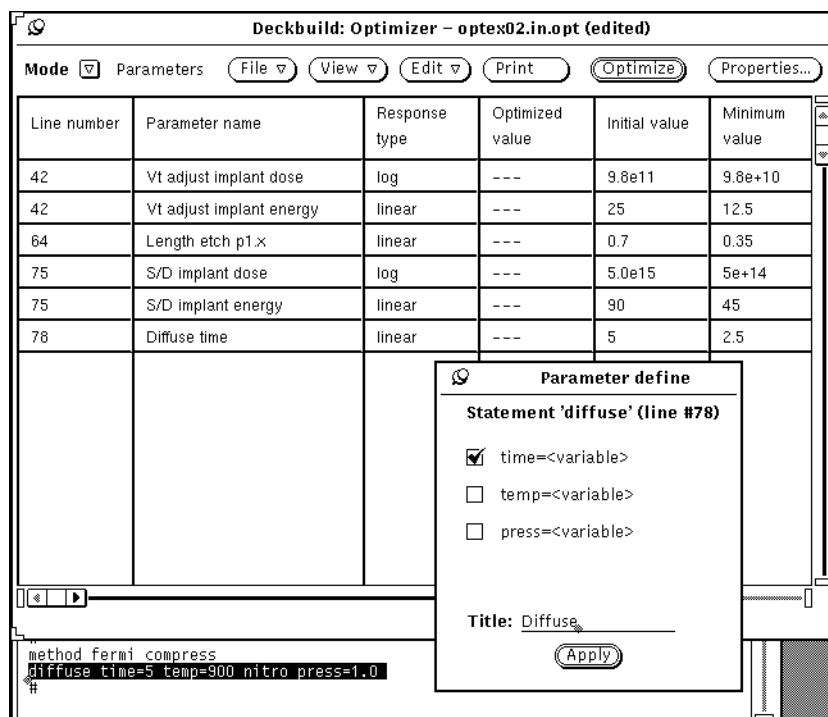


Figure 6-6: Parameters Added to Worksheet

Repeat the process to define parameters on the other lines in the input deck. After adding each parameter, to change its initial value, minimum allowed value, or maximum allowed value. See Section 6.3.3: “Editing A Parameter” for more information.

6.3.2: Deleting Parameters

To delete a parameter:

1. Place the pointer anywhere in the row to be deleted and double-click the **SELECT** mouse button to make a row selection. The row then appears raised (Figure 6-7).
2. Choose **Delete** from the pull-down **Edit** menu to remove the selected row from the worksheet.

The **Delete** operation works on all currently selected rows. More than one row at a time can be deleted. See Section 6.8: “Worksheet Editing” for instruction on selecting more than one row.

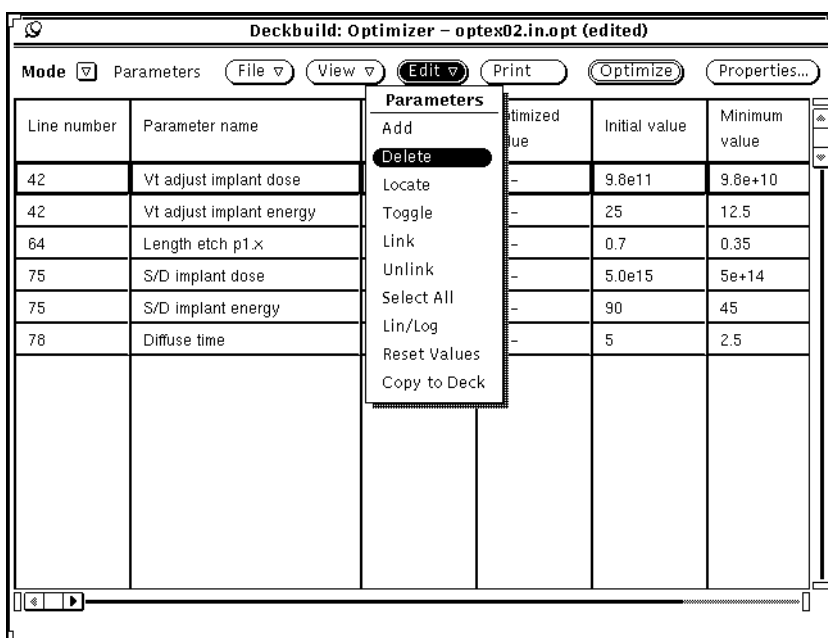


Figure 6-7: Deleting a Parameter

6.3.3: Editing A Parameter

The **Parameter** worksheet allows direct numeric cell value editing, such as **Initial value**, **Minimum value**, and **Maximum value**. See “Parameter Fields” on page 6-11 for an explanation of the fields.

Editing Numeric Values

To edit a numeric cell value:

1. Position the pointer over the cell. The cell will appear indented.
2. Click SELECT once and a text caret appears in the cell.
3. Edit the cell value. The worksheet understands normal keyboard input plus Control-U, which erases the cell contents. Enter the new value, replacing the existing value.
4. Finish the edit by pressing the **Return** key. The new value remains on the worksheet cell.

When the cell is read-only and cannot be edited, a warning message is displayed in the lower left corner of the worksheet. See Section 6.8: “Worksheet Editing” for more information on using the worksheet.

Editing The Response Type

Response type is either **linear** or **log**. Response type should be set to **log** if the extracted results vary logarithmically with the input parameter.

To change the response type:

1. Position the pointer anywhere in the row and double-click the SELECT mouse button to capture the row. The row appears raised.
2. Choose **Lin/Log** from the **Edit** pull-down menu. The selected row’s response type are toggled.

Since **Lin/Log** operates on all selected rows, you can select and toggle multiple rows at the same time.

Editing The Parameter Name

The OPTIMIZER forms the parameter names when entered by appending the parameter name (shown next to the checkboxes on the **Parameter define** popup) with the title field on the popup. By default, the title is the command word (first word) on the line. For example, the parameter dose in an implant statement would be called **implant dose** on the worksheet. If you type LDD Implant as the title, the parameter name will be called **LDD Implant dose** on the worksheet.

To help distinguish the parameter names, you may need to change them once entered on the worksheet. To change a parameter name:

1. Choose **View→Control...** (Figure 6-8) and the **Parameter control** popup will appear (Figure 6-9).

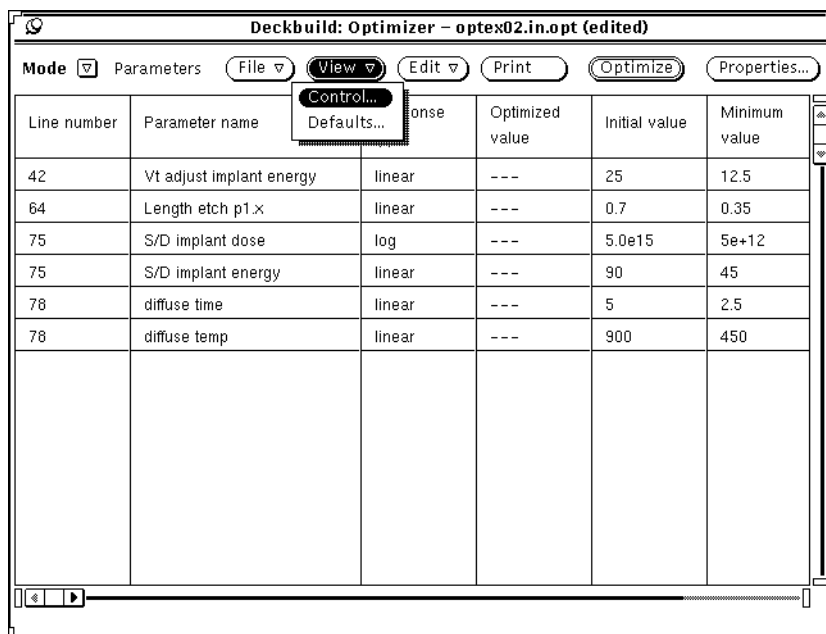


Figure 6-8: Parameter View Menu

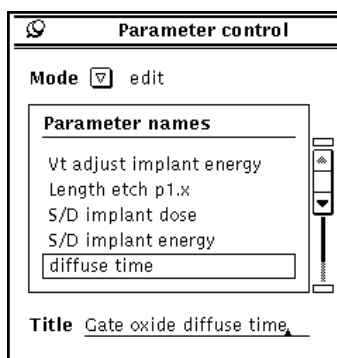


Figure 6-9: Parameter Control Popup, Edit Mode

2. Change **Mode** to **edit** on this popup. The scrolling list contains all the parameter names.
3. Click **SELECT** over the parameter name that needs to be changed. The parameter name appears under **Title**. Enter in the new parameter name under **Title** and press the Return key.

Once you press Return, the parameter name will be updated on the worksheet and on the scrolling list.

6.3.4: Linked Parameters

Occasionally, it is desirable to “link” two or more parameters together. For instance, a series of several diffusion steps in a row might all use the same HCl percentage or the same pressure. When they are entered as parameters but not linked, each parameter is varied independently during the optimization run. This is not meaningful if the physical diffusions are done in the same environment. When parameters are linked together, they all vary together as one during the optimization run.

Do the following to link parameters together.

1. Select a row to be linked by double-clicking **SELECT** over that row.
2. Select one or more additional rows by single-clicking **ADJUST** over each additional row.
3. Choose **Link** from the **Edit** pulldown menu. **Link** applies to all currently selected rows.

The **Link** operation works by referring the second, third and other linked rows back to the first selected row (counting from the top down). That first row becomes the master row to which all the others are linked (Figure 6-10).

Line number	Parameter name	Response type	Optimized value	Initial value	Minimum value
78	Pressure 1	linear	---	1.00	0.5
79	Drive time	linear	---	60	30
79	Pressure 2->78:Pressure 1	linear	---	1.00	0.5
80	Pressure 3->78:Pressure 1	linear	---	1.00	0.5

```
#
method fermi compress
diffus time=10 temp=900 t.final=1000 nitro press=1.00
diffus time=60 temp=1000 nitro press=1.00
diffus time=10 temp=1000 t.final=900 nitro press=1.00
#
```

Figure 6-10: Linked Parameters

After all rows have been linked, the linked parameter titles change. The titles are appended with an arrow, the line number, and the master parameter name. For example, Pressure 2 becomes Pressure2->78:Pressure 1.

Linked parameters all share the same response type and optimized, initial, minimum, and maximum values. The initial, minimum, and maximum values are taken from the first, parent parameter, but are not updated on the child parameters. This preserves their independent settings in case the parameters are unlinked later. The optimized value is taken from the master parameter and updated on each of the linked parameters during optimization.

It is also possible to link in a new parameter to a set of already-linked parameters. Select any one of the already linked parameters, plus the new parameter, and choose **Link** again on the menu. The OPTIMIZER knows how to follow the sequence of links.

If the master parameter is deleted, all links to it are automatically removed. If a linked parameter is deleted, it does not affect any of the other links. It is important to remember that links are not saved when the optimizer setup is stored to a file. The links have to be set up again if an optimizer file is reloaded. To unlink a parameter:

1. Select the row to be unlinked by double-clicking **SELECT**.
2. Select additional rows, if desired, by single-clicking **ADJUST**.
3. Choose **Unlink** from the **Edit** menu.

6.3.5: Parameter Defaults

The **Parameter** worksheet has built-in values to determine the appropriate response type and minimum and maximum values for a new parameter.

By default, parameters are of response enter log, if their initial value is above $1e+10$ and min/max values are $\pm 50\%$ for linear parameters, and ± 1 decade for log parameters.

You can change response type and min/max values after adding parameters. These defaults only help make adding parameters quicker and easier.

Do the following to change the default response type and min/max ranges used when adding a new parameter.

1. Select **View→Defaults...** and the **Parameter defaults** popup will appear (Figure 6-11).

Figure 6-11: Parameter Defaults

2. Enter the desired new default value(s).
3. Click on **Apply** when finished. If you want to save the values as permanent defaults, click on **Save**.

6.3.6: Copying Parameters To The Deck

The OPTIMIZER automatically fills in the **Optimized value** column on the **Parameter** worksheet as it runs. If the OPTIMIZER converges successfully and retaining the optimized parameter values in the input deck is desired, select **Edit→Copy to Deck**.

Copy to Deck copies all optimized parameter values back into the deck in place of the former values. Save the input deck (using the **DeckBuild File** menu) to generate the permanent changes.

6.3.7: Enabling/Disabling Parameters

The OPTIMIZER allows the disabling of parameters, which are then ignored during optimizations. Disabling is often useful when a large number of parameters have been entered. But, you may want to try freezing one or more at certain values without deleting them from the worksheet. To disable a parameter:

1. Position the pointer anywhere on the row to be disabled and double-click the **SELECT** mouse button to select that row. The row is selected.
2. Choose **Toggle** from the **Edit** pull-down menu. The selected row is grayed out on the worksheet, and its values are frozen.

To enable a disabled parameter, follow the same procedure starting with a disabled row.

Note: During the optimization run, the OPTIMIZER uses whatever entered optimized value for disabled parameters. If no optimized value exists, the initial value is used.

6.3.8: Folding Columns

The OPTIMIZER allows folding, or hiding, any column or columns on the worksheet. This can be useful when several columns on opposite sides of the worksheet need to be seen simultaneously, without having to scroll horizontally back and forth. To fold a worksheet column:

1. Choose **Control...** from the View pulldown menu, and the **Parameter control** popup shown in Figure 6-12 appears.

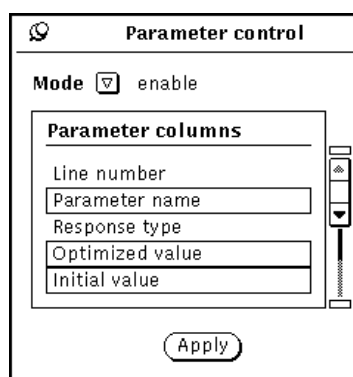


Figure 6-12: Parameter Control Popup, Enable Mode

2. Change **Mode** to **enable** on this popup.
3. Click **SELECT** on any columns in the scrolling list that you want to fold. Columns that remain selected on the scrolling list are shown; the others are folded and are not shown. Click on **Apply** to apply the changes.

Parameter Fields

- **Line number** — The line number in the input deck on which the parameter is used. The line number is updated automatically when the input deck is edited.
- **Parameter name** — The name of the parameter. The name can be redefined from the **Parameter** control popup.
- **Response type** — Either **linear** or **log**. The response type determines how far the OPTIMIZER swings the parameter value. Log type parameters are swung further than linear parameters, and should be used for parameters which have a logarithmic effect on the extracted results.
- **Optimized value** — The value determined by the OPTIMIZER to achieve the best fit to the target(s).
- **Initial value** — The initial value of the parameter, taken from the input deck.
- **Minimum value** — The minimum value that the OPTIMIZER is allowed to use.
- **Maximum value** — The maximum value that the OPTIMIZER is allowed to use.

Note: As an aid to optimization, the Optimizer uses Optimized value as each input parameter's initial value after the Optimizer has been run once. To use the Initial value once again, choose Reset Values on the Parameter Edit menu. The optimized values are deleted on the Parameter worksheet.

6.4: Targets

A target is the value of any valid extract statement in the input deck. **EXTRACT** statements are special commands understood by **DECKBUILD** that extract certain properties of the simulated device. For example, you can extract properties, such as material thickness, net doping versus depth, and junction depth for process simulation and extract I-V curves and parameterized values, such as V_t , Θ , and $DIBL$ for device simulation. **EXTRACT** also allows you to construct your own routine to perform customized manipulation of points and curves.

DECKBUILD provides full popup-driven generation of the extract statements. For more information, see Chapter 5: “Extract”.

For purposes of the **OPTIMIZER**, **EXTRACT** statements come in two flavors. Those that extract a single value (like V_t) called point targets, and those that extract an entire curve (like V_g vs. I_d) called curved targets.

6.4.1: Adding A Target

To add a target, follow these four steps:

1. Display the **Target** worksheet by setting **Mode** to **Targets**. To do this, position the pointer over **Mode**, press the **MENU** mouse button, and select **Targets**. The **Target** worksheet will then appear (Figure 6-13).

Line number	Target name	Target type	X value	Target value	Optimized value
130	Vt curve	log	0	9.10474e-14	---
130		log	0.5	5.94327e-09	---
130		linear	1	3.6632e-06	---
130		linear	1.5	9.51657e-06	---
130		linear	2	1.42802e-05	---
130		linear	2.5	1.82314e-05	---
130		linear	3	2.16368e-05	---
130		linear	3.5	2.46356e-05	---
130		linear	4	2.72568e-05	---
130		linear	4.5	2.9631e-05	---
130		linear	5	3.17689e-05	---

Figure 6-13: Target Worksheet

2. Select (highlight) the line in the input deck that contains the target you want to add. To do this, position the pointer over the line, triple-click the **SELECT** mouse button to capture the entire line. The **OPTIMIZER** also accepts lines that have only a word or any character(s) selected.
3. Choose **Add** from the **Edit** pull-down menu. To do this, move the pointer over the **Edit** button, click **MENU** and select **Add** (Figure 6-14).

A new row is inserted into the **Target** worksheet for the selected extract statement (Figure 6-15). The rows are always ordered by line number.

4. Enter a target value for the new target. Target value entry depends on the extract value type (point or curve). Both types of target value methods are explained on the following pages.

The screenshot shows the Deckbuild V3.1.8 Alpha Optimizer interface. The main window is titled "Deckbuild: Optimizer - optex02.in.opt (edited)". It features a menu bar with options: File, View, Edit, Find, Main Control, Commands, and Tools. Below the menu bar is a toolbar with buttons for Mode, Targets, File, View, Edit, Print, Optimize, and Properties... The Optimize button is highlighted.

The central part of the interface is a table with the following columns: Line number, Target name, Target type, X value, Target value, and Optimized value. The table contains 13 rows of data, all with a line number of 130. The first row has a target name of "vt" and a target value of "???". The subsequent rows have target names of "Vt curve" and "vtcurve", and target values ranging from 9.10474e-14 to 3.17689e-05.

Below the table is a code editor with the following text:

```
make-iv
*
end_error=0.01000
extract name="vt" (xintercept(maxslope(curve(abs(vg),abs(id)))) - abs(vd)/2.0)
extract name="vtcurve" curve(abs(vg),abs(id)) outf="vtcurve"
```

The bottom right corner of the window displays the text "SSUPREM3".

Figure 6-14: Targets Added to Worksheet

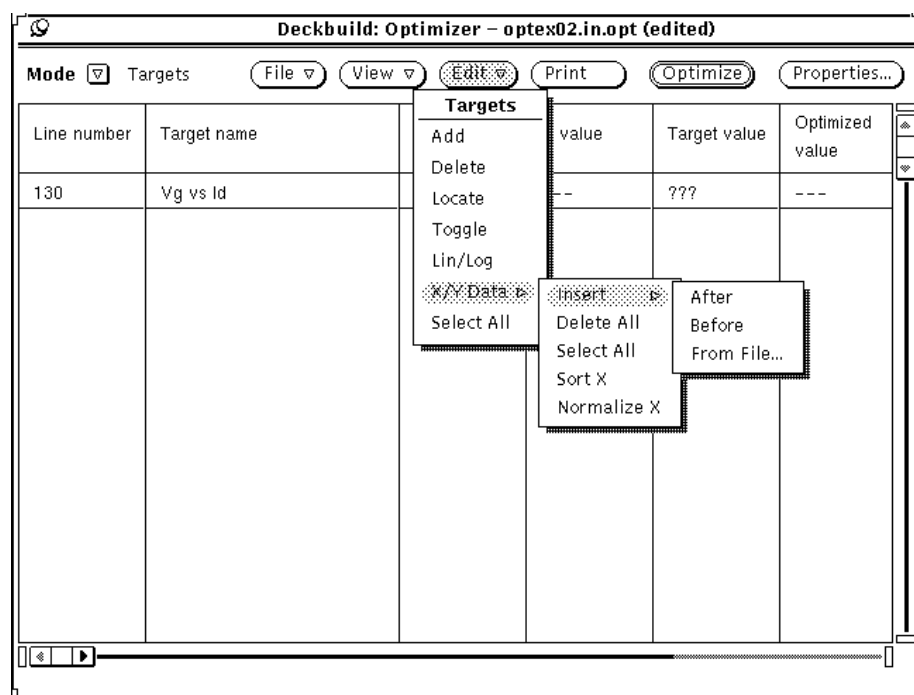


Figure 6-15: Target X/Y Data Menu

Defining a Point Value Target

For point targets, enter the target value directly into the **Target value** column. See Section 6.4.3: “Editing A Target” for more information. Repeat this process to define other targets in the input deck.

Defining a Curve Value Target

For curved targets, you must enter one or more x/y coordinate pairs. The **X value** column on the worksheet corresponds to the x axis of the extracted curve. The **Target value** column corresponds to the y axis. You can do any of the following to enter x/y coordinate pairs for curved targets.

- Insert pairs into the worksheet,
- Read in x/y data from a TONYPLOT data format file
- Do a combination of both.

Creating A Curved Target From Scratch

Do the following to insert an x/y value pair.

1. Position the pointer anywhere in the target row and double-click the SELECT mouse button to select that row. The row then appears raised.
2. Choose **Edit**→**X/Y Data**→**After** (Figure 6-15). A new row will be inserted after the selected row. You can insert additional data rows either before or after any other data row.

Note: Only the first row of a curved target shows the target name (Figure 6-16).

3. Enter **X value** and **Target value** values for the new row.

Repeat this process to define the entire curved target.

The screenshot shows the 'Deckbuild: Optimizer - optex02.in.opt (edited)' window. It features a menu bar with 'Mode', 'Targets', 'File', 'View', 'Edit', 'Print', 'Optimize', and 'Properties...'. Below the menu is a table with the following data:

Line number	Target name	Target type	X value	Target value	Optimized value
130	Vg vs Id	linear	---	???	---
130		linear	---	???	---
130		linear	---	???	---
130		linear	---	???	---

Figure 6-16: A Curved Target

Creating a Curved Target From a Data File

When a TONYPLOT data format file is on the disk that you want to use as an optimization target, the OPTIMIZER allows it to be read directly into the worksheet.

To create a curved target from a data format file, follow these steps:

1. Position the pointer anywhere in the target row, and double-click the **SELECT** mouse button to select that row. The row will then appear raised.
2. Choose **Edit**→**X/Y Data**→**From File...** and the **Optimizer X/Y Target Data** popup will appear (Figure 6-17).
3. Enter in the directory and filter or double-click on directory names in the scrolling list to change directory. To load a file, double-click on the file name in the scrolling list or select the file name in the list and click on **Load**. If the file is in the correct format, existing data rows for the curved target are removed and replaced by rows constructed from the X/Y data in the data file.

To quickly and easily create a TONYPLOT data format file for the target, specify a file name on the extract statement. For example:

```
extract curve(depth, abs(net)) outf="net_doping.dat"
```

EXTRACT saves the file in TONYPLOT data format. Run the deck through using DECKBUILD, then load in the saved data file into the worksheet, add/delete rows and edit the values to create the desired target curve.

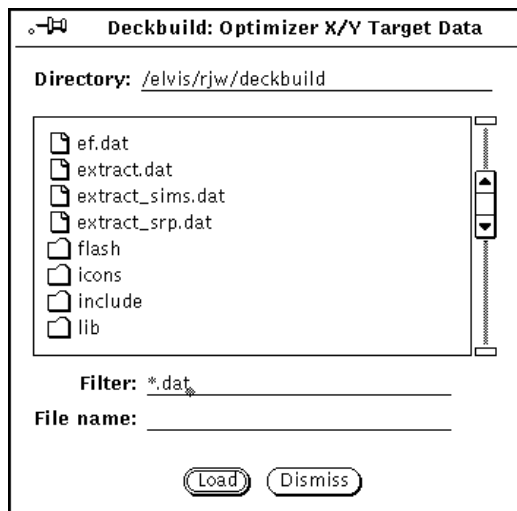


Figure 6-17: X/Y Target Data Popup

6.4.2: Deleting A Target

To delete a target:

1. Position the pointer anywhere in the row to be deleted and double-click the **SELECT** mouse button to select that row. The row will then appear raised (Figure 6-18).
2. Choose **Delete** from the **Edit** pull-down menu. The selected row will be removed from the worksheet.

The **Delete** operation works on all currently selected rows, more than one row at a time can be deleted. See Section 6.8: “Worksheet Editing” for procedures on selecting more than one row.

For curved targets, **Delete** deletes only the selected x/y data point(s). To delete the entire target at once, select any row in the curve to be deleted (as in step 1 above) and choose **Delete All** from the **X/Y Data** menu (Figure 6-19).

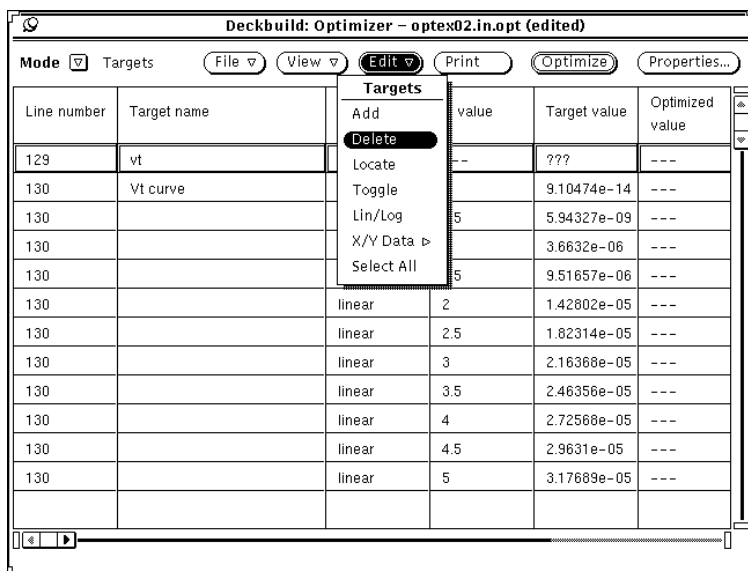


Figure 6-18: Deleting a Target

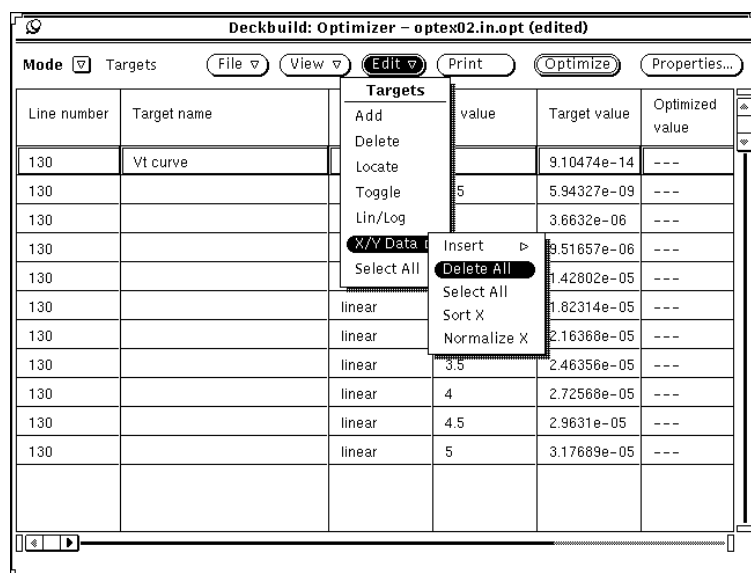


Figure 6-19: Deleting a Curved Target

6.4.3: Editing A Target

The **Target** worksheet allows direct edit of numeric cell values, such as **X value**, **Target value**, and **Weight**. See “Target Fields” on page 6-19 below for an explanation of the fields.

Editing Numeric Values

To edit a numeric cell value:

1. Position the pointer over the cell. The cell then appears indented.
2. Click SELECT once. A text caret appears in the cell.
3. Edit the cell value. The worksheet understands normal keyboard input plus Control-U, which erases the cell contents. Enter the new value.
4. Finish the edit by pressing the Return key. The new value remains in the worksheet cell.

If the cell is read-only and cannot be edited, a warning message is displayed in the lower left-hand corner of the worksheet.

See Section 6.8: “Worksheet Editing” for more information about using the worksheet.

Editing the Target Type

Target type is either **linear** or **log**. Set the response type to **log** if the target value is extremely small to reduce the target’s apparent error. For extremely small targets, a very small absolute difference between the target and the extracted data could result in an extremely large percentage error (1e-12 vs. 1e-11 is a 900% error). Setting target type to **log** tells the OPTIMIZER to take the log of the result and target before computing the percentage error.

To change the target type:

1. Position the pointer anywhere in the row and double-click the SELECT mouse button to capture that row. The row will appear raised.
2. Choose **Lin/Log** from the **Edit** pulldown menu. The selected row’s response type is toggled.

Since **Lin/Log** operates on all selected rows, you can select multiple rows and toggle at the same time.

Editing The Target Name

The OPTIMIZER fills in the target name, by default, when it is first added to the worksheet. The target name is taken from the **name** token in the extract statement. You may want to change the target names to help distinguish them. To change a target name:

1. Choose **Control...** from the **View** pulldown menu and the **Target control** popup will appear (Figure 6-20).

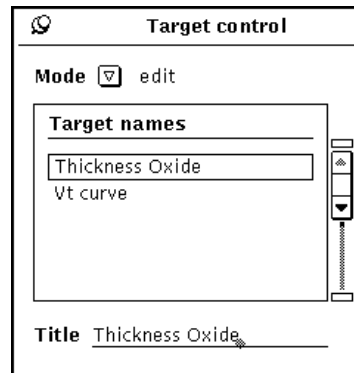


Figure 6-20: Target Control Popup - Edit Mode

2. Change **Mode** to **edit** on this popup. The scrolling list contains all the target names.
3. Click **SELECT** over the target name to be changed. The target name appears under **Title**. Enter the new target name under **Title** and press Return.

When you press Return, the target name will be updated on the worksheet and on the scrolling list.

6.4.4: Enabling/Disabling Target

The OPTIMIZER allows you to disable targets. Disabled targets are ignored during the optimization. Disabling is often useful when a large number of targets have been entered. There is a need to concentrate on an important subset of targets without deleting the remaining targets from the worksheet.

To disable a target:

1. Position the pointer anywhere in the row that needs to be disabled and double-click the **SELECT** mouse button on the row. The row is selected.
2. Choose **Toggle** from the **Edit** pulldown menu.

The selected row is grayed out on the worksheet, and its values are be frozen. To enable a disabled target, follow the same procedure precisely beginning with a disabled row. During an optimization run, the OPTIMIZER does not monitor or count the error terms from disabled targets.

6.4.5: Folding Columns

The OPTIMIZER allows you to fold, or hide, any column or columns on the worksheet. This may be useful when there is a need to see several columns on opposite sides of the worksheet without having to horizontally scroll back and forth.

To fold a worksheet column:

1. Choose **Control...** from the **View** pulldown menu and the **Target control** popup will appear (Figure 6-21).

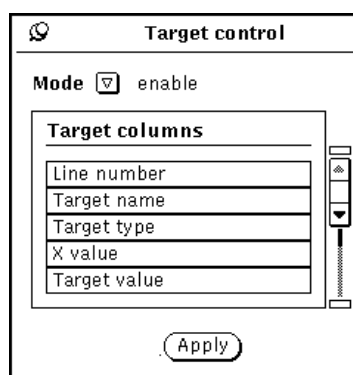


Figure 6-21: Target Control Popup - Enable Mode

2. Change **Mode** to **enable** on this popup.
3. Click **SELECT** on any columns that are to be folded in the scrolling list. Columns remaining selected on the scrolling list are shown. The others are folded and not shown. Press **Apply** to apply changes.

Target Fields

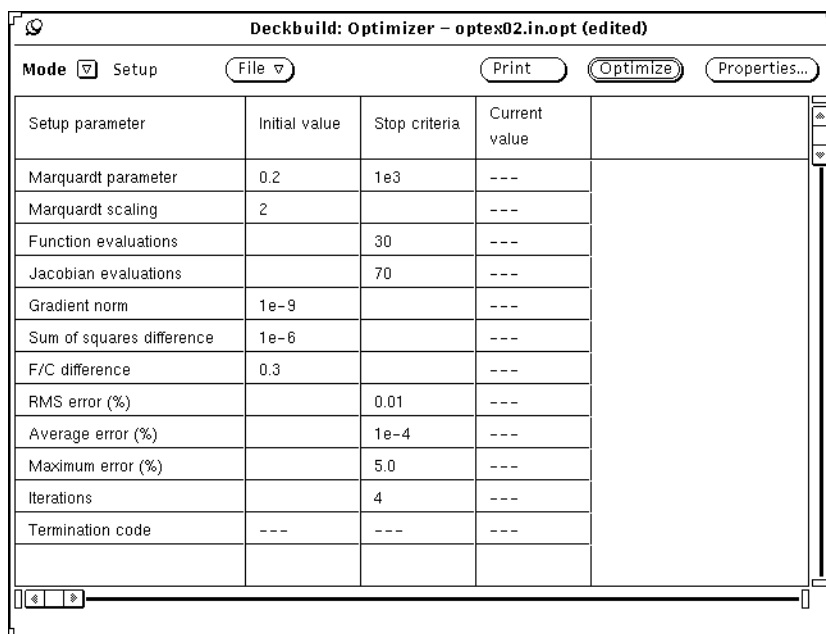
- **Line number** — The line number that the target is on in the input deck. The line number is automatically updated when the input deck is edited.
- **Target name** — The name of the target. You can redefine the name using the **Target control** popup.
- **Target type** — Either **linear** or **log**. The target type determines how the **Optimizer** computes the error term. Log type targets should be used where the target value is extremely small.
- **X value** — The X axis value of an x/y coordinate pair. It is only used by curved targets.
- **Target value** — The optimization target value (Y axis value for curved targets).
- **Optimized value** — The current value of the extracted (simulated) data.
- **Error (%)** — The percentage error between **Target value** and **Optimized value**.
- **Weight** — The target weighting factor. The weight is multiplied by the actual error to determine the apparent error. A value of 1 indicates that the **Optimizer** shall use the actual error, a value of 0.5 indicates that it should use only half the actual error. Values between 0 and 1 make the target less sensitive. Therefore, it is subject to wider error tolerance, while values greater than 1 make it more sensitive and force the simulated data to smaller error tolerance. For instance, a weight of 2.0 with a setup maximum error of 5.0% would force a target to have a maximum actual error of 2.5% to achieve convergence.

Note: The error percentage shown on the Targets worksheet is the actual error percentage before weighting.

6.5: Setup

6.5.1: Overview

The **Setup** worksheet controls a number of constants used by the OPTIMIZER, such as the maximum number of iterations and evaluations and several different convergence criteria. The default values are almost always adequate. The **Setup** worksheet is displayed by setting **Mode** to **Setup** (Figure 6-22).



Setup parameter	Initial value	Stop criteria	Current value
Marquardt parameter	0.2	1e3	---
Marquardt scaling	2		---
Function evaluations		30	---
Jacobian evaluations		70	---
Gradient norm	1e-9		---
Sum of squares difference	1e-6		---
F/C difference	0.3		---
RMS error (%)		0.01	---
Average error (%)		1e-4	---
Maximum error (%)		5.0	---
Iterations		4	---
Termination code	---	---	---

Figure 6-22: Setup Worksheet

The most important setup criteria are as follows:

- **Maximum error** — Determines the maximum allowable error for any target to achieve convergence.
- **Iterations** — Determines how many total Marquardt iterations the optimizer is allowed to perform. This value counts only the Marquardt iterations, each consists of one or several sub-iterations. The actual number of simulation calculations is always greater than this value.

The default value of **Maximum error** and **Iterations** is 5.0% and 4 respectively.

6.5.2: Editing Setup Values

To edit a setup cell value:

1. Position the pointer over the cell. The cell then appears indented.
2. Click SELECT once. A text caret appears in the cell.
3. Edit the cell value. The worksheet understands normal keyboard input plus Control-U, which erases the cell contents. Enter the new value.
4. Finish the edit by pressing Return. The new value remains in the worksheet cell.

When the cell is in read-only, a warning message is displayed in the lower left-hand corner of the worksheet. See Section 6.8: “Worksheet Editing” for more information on using the worksheet.

6.5.3: Saving The Setup

To modify the setup data and to save for future runs, choose **Save Setup** from the **File** menu. The setup information is saved in \$HOME/.masterrc.

6.6: Graphics

6.6.1: Overview

The OPTIMIZER maintains a run-time graphics visualization system that allows for tracking the progress of the OPTIMIZER over its iterations, and to evaluate its current status. To access graphics, set the **Mode** to **Graphics** (see Figure 6-23).

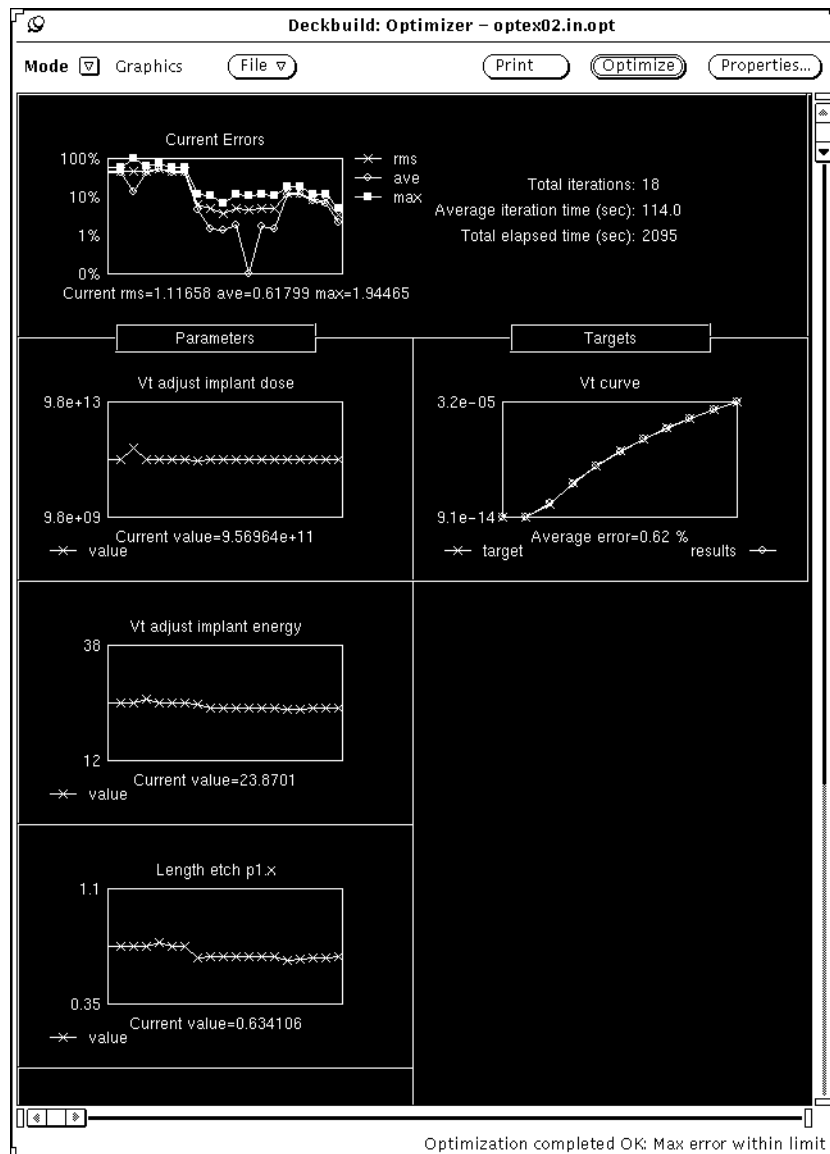


Figure 6-23: A Curved Target

The **Current Errors** section at the top of the window maintains the values of RMS error, average error, and maximum error over all targets. It also displays the total number of iterations so far, the average iteration time, and the total elapsed time. The total elapsed time increments in real time as the optimization runs. The number of iterations counts the number of simulation computations, not Marquardt iterations defined on the **Setup** worksheet.

The **Parameters** column keeps track of the input parameter values. Each rectangle represents, vertically, the allowable range of the parameter (its minimum to maximum value), while the horizontal axis represents the number of iterations. The parameter values are displayed for all iterations to date.

The **Targets** column shows the target values, the current simulated values, and an error term. For single-point targets, the target is drawn as a horizontal line, and simulated values plus error terms are drawn for all iterations to date. For curved targets, the target curve and the simulated results curve are plotted plus the current average error over the entire curve.

This **Graphic** iteration history information is maintained until another optimization run is started or until any target or parameter is added or deleted.

6.7: Results

6.7.1: Overview

An optimized results list is maintained by the OPTIMIZER to show the input parameter values and target results for each iteration. To display the **Results** worksheet, set the **Optimizer's Mode** to **Results**. Figure 6-24 shows a typical **Results** worksheet.

Iteration	Vt adjust implant dose	Vt adjust implant energy	Length etch p1.x	S/D implant dose	S/D implant energy	Vt curve
1	9.8e+11	25	0.7	5e+15	90	err=32%
2	2.34651e+12	25	0.7	5e+15	90	err=7.3%
3	9.8e+11	25.7906	0.7	5e+15	90	err=34%
4	9.8e+11	25	0.722136	5e+15	90	err=42%
5	9.8e+11	25	0.7	1.56825e+16	90	err=32%
6	9.8e+11	25	0.7	5e+15	92.846	err=32%
7	9.22232e+11	24.6085	0.632424	2.36997e+13	86.322	err=1.7%
8	9.32321e+11	23.9048	0.636331	5e+12	83.6261	err=0.36%
9	9.46142e+11	23.9048	0.636331	5e+12	83.6261	err=0.31%
10	9.32321e+11	23.9839	0.636331	5e+12	83.6261	err=0.47%
11	9.32321e+11	23.9048	0.638544	5e+12	83.6261	err=0.0094%
12	9.32321e+11	23.9048	0.636331	5.6055e+12	83.6261	err=0.45%
13	9.32321e+11	23.9048	0.636331	5e+12	83.9107	err=0.35%

Optimization completed OK: Max error within limit

Figure 6-24: Results Worksheet

The left column displays the iteration number, starting from 1. The other columns display the input parameters from the **Parameters** worksheet followed by targets from the **Targets** worksheet.

Input parameter columns show the input parameter values used for each iteration. Target columns show the extract values obtained for the parameter values, or if a curved target, the average error for the entire curve.

The **Results** worksheet is automatically updated at run-time and a new row is added every time an iteration finishes. If any of the parameter or target names are changed, the changes are reflected in the **Results** worksheet.

6.7.2: Sensitivity

The **Results** screen also displays a sensitivity calculation performed as the optimization progresses.

The sensitivity of each target is displayed next to the target's value column. The sensitivity is calculated as the percentage change in the target value divided by the percentage change in a single input value.

The sensitivity displayed on a given row is the result of changing the single input parameter on that row, which differs from the baseline calculation (the topmost result row).

Since the OPTIMIZER first calculates the baseline values and performs the sensitivity calculation on each parameter in turn, thereafter trying to vary all parameters as necessary, the sensitivity results are displayed only on the second through *n*th row of the results worksheet, where *n* is the number of parameters plus one.

6.8: Worksheet Editing

6.8.1: Overview

The **Worksheet** allows for editing cells and typing values directly into the cell contents. Some cells, however, (depending on the worksheet mode) are read-only and cannot be edited.

6.8.2: Numeric Values

To edit a numeric cell value on a worksheet:

1. Position the pointer over the cell. The cell appears indented.
2. Click **SELECT** once. A text caret appears in the cell (Figure 6-25).
3. Edit the cell value. The worksheet understands normal keyboard input, plus Control-U, which erases the cell contents. Enter the new value.
4. Finish the edit by pressing Return. The new value remains in the worksheet cell.

When the cell is read-only, a message footer is displayed in the lower left corner of the worksheet.

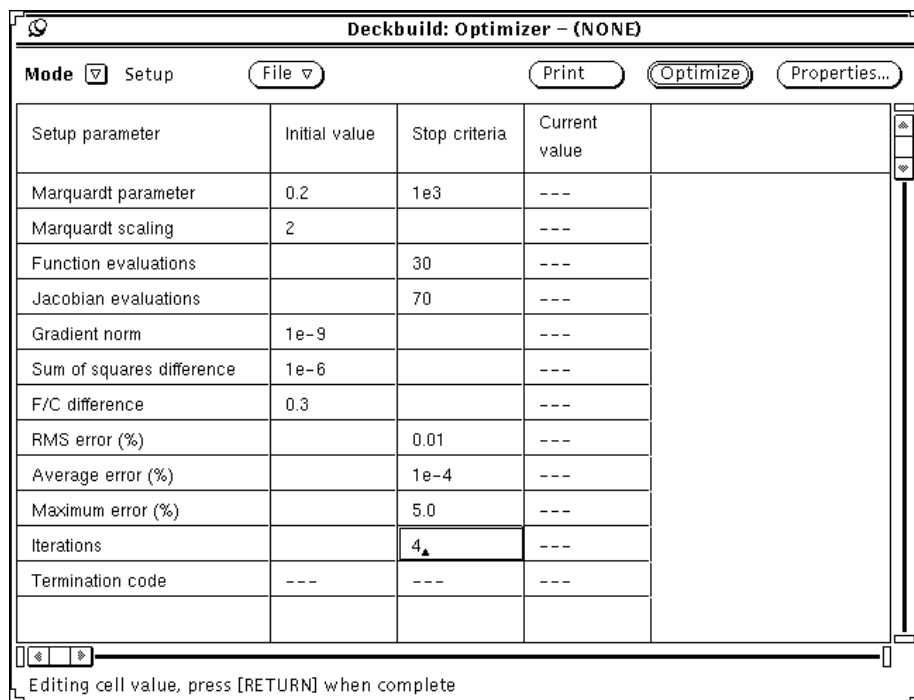


Figure 6-25: Editing a Worksheet Cell

6.8.3: Selecting Rows

To select a worksheet row, position the pointer anywhere over the row and double-click the **SELECT** mouse button. To select a second or any further row (extending the range of the selection), click **ADJUST** over each additional row to be selected. To deselect a row, position the pointer over the selected row and click **ADJUST**. **ADJUST** toggles the selected state. To deselect all rows, press **SELECT** once anywhere in the worksheet.

Note: Selections are lost when the worksheet mode is changed.

6.8.4: Mouseless Operation

The worksheet supports mouseless operation. You can traverse rows and columns of the worksheet with the arrow keys, plus the Home, End, PgUp, and PgDn keys. You can edit a worksheet by pressing Return in a cell.

6.9: File I/O

6.9.1: Overview

When an optimization run is finished, the OPTIMIZER allows storage of all pertinent data to disk. You can load an optimization file at any later point to initialize the OPTIMIZER with all the saved target and parameter data. The Setup data is stored independently. For procedures on Setup data, see Section 6.5: “Setup”.

6.9.2: Creating A New File

To create a new optimization file:

1. Move the pointer to the **File** menu button, press the MENU mouse button, and select **Store as New File...** (Figure 6-26). The **Store** popup will then appear (Figure 6-27).

Note: The current directory is listed on the Store popup.

2. Move the pointer into the **Store** window and enter the name of the preferred file. By default, the file name is the name of the current input deck with .opt appended.
3. Click on **File**→**Store as New File** and an ASCII format optimization file will be saved. This file contains all parameter and target information.

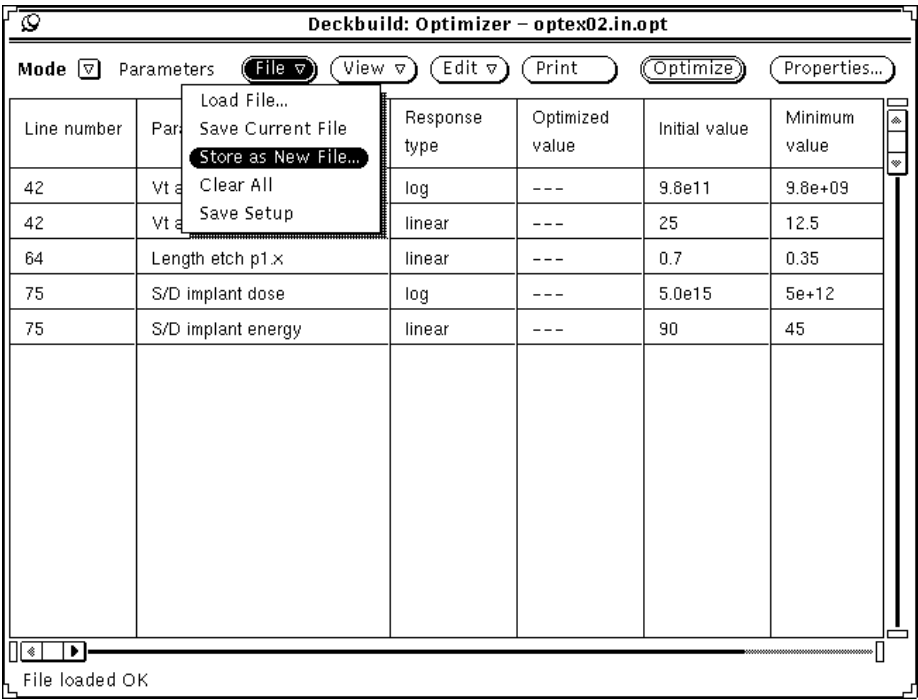


Figure 6-26: Storing a New File

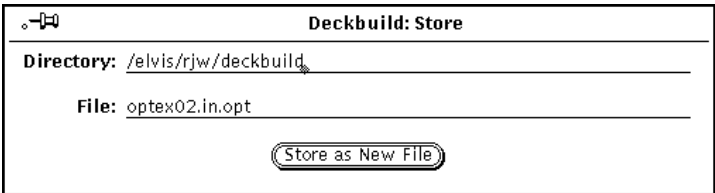


Figure 6-27: Store Popup

Note: Parameter links are not saved in the optimization file. Any linked parameters have to be relinked when the optimization file is loaded.

6.9.3: Working With An Existing File

Any previously saved optimization file can be loaded and modified or stored under another name to create a new file, leaving the original intact.

To load a file into the OPTIMIZER:

1. Choose **File→Load File...** on the **Optimizer** window and the Optimizer Load Popup shown in Figure 6-28 will appear.

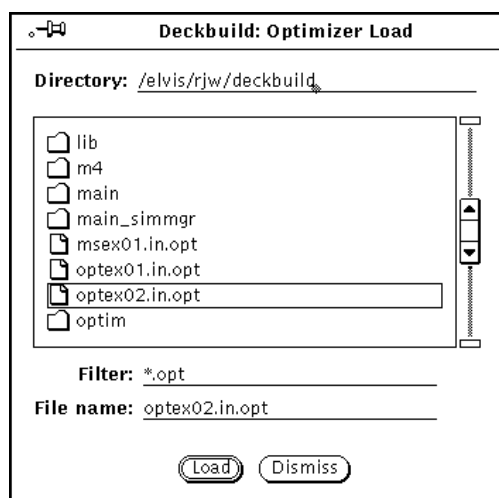


Figure 6-28: Optimizer Load Popup

2. Adjust **Filter**, if necessary, to filter out all but the file names of interest. By default, the filter is *.opt. Press Return to apply the filter.
3. Change the directory, if necessary, by double-clicking on a directory in the scrolling list or by entering a directory name under **Directory**.
4. Once you have found the file you want to load, load it by double-clicking on it in the scrolling list. You can also load the file by clicking on **Load**.

Note: The file to be loaded must correspond to the current input deck. If the information in the optimizer file does not match the contents of the input deck, the OPTIMIZER will refuse to load the file and an error message appears. For this reason, always save the input deck and its optimizer file at the same time.

6.9.4: Saving The Existing File

You can now edit the OPTIMIZER data and save those changes back to the existing OPTIMIZER file or save them as a new file.

To save the changes to the existing file, choose **File→Save Current File** on the OPTIMIZER window.

Be sure to save the input deck as well, if it has been edited. Save the input deck using DECKBUILD's **File** menu.

6.10: Printing

6.10.1: Overview

To print any of the worksheets or graphics data:

1. Press **Properties...** on the **Optimizer** window and the **Optimizer Properties** popup will appear. Change the **Category** in the upper left corner to **Graphics printer** (Figure 6-29).
2. Select the **Destination**, **Type**, **Layout**, and **Page** size of the output. If the destination is **Printer**, also choose the printer queue.

By default, the output will go to a file called `print.out` in PostScript format. Indexed, enabled by default, means a unique file name `print.out` is made each time a print is done, so previous files are never overwritten. Click on **Save** to permanently save the settings.

3. Press **Print** back on the **Optimizer** window. The output goes to the file or printer specified in Step 2 and contains data corresponding to the current **Optimizer** mode.

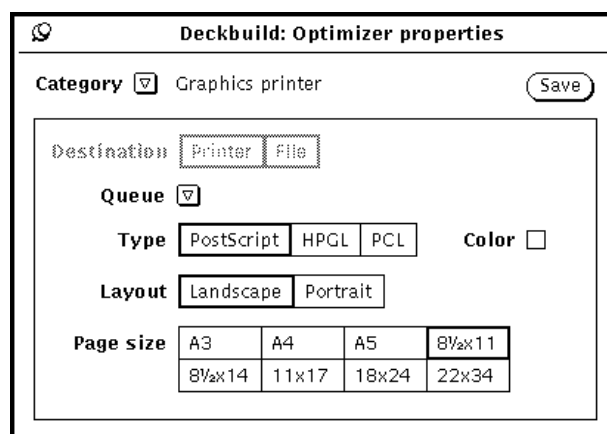


Figure 6-29: Optimizer Properties Popup - Graphic Printer Category

Figures 6-30 and 6-31 shows examples of printed output.

	1	2	3	4
Line number	42	42	64	75
Parameter name	Vt adjust implant dose	Vt adjust implant ener,	Length etch pl.x	S/D implant dose
Response type	log	linear	linear	log
Optimized value	9.56964e+11	23.8701	0.634106	5.51201e+12
Initial value	9.8e11	25	0.7	5.0e15
Minimum value	9.8e+09	12.5	0.35	5e+12
Maximum value	9.8e+13	37.5	1.05	5e+17

	5
Line number	75
Parameter name	S/D implant energy
Response type	linear
Optimized value	83.3429
Initial value	90
Minimum value	45
Maximum value	160

Figure 6-30: Parameters Worksheet Printout

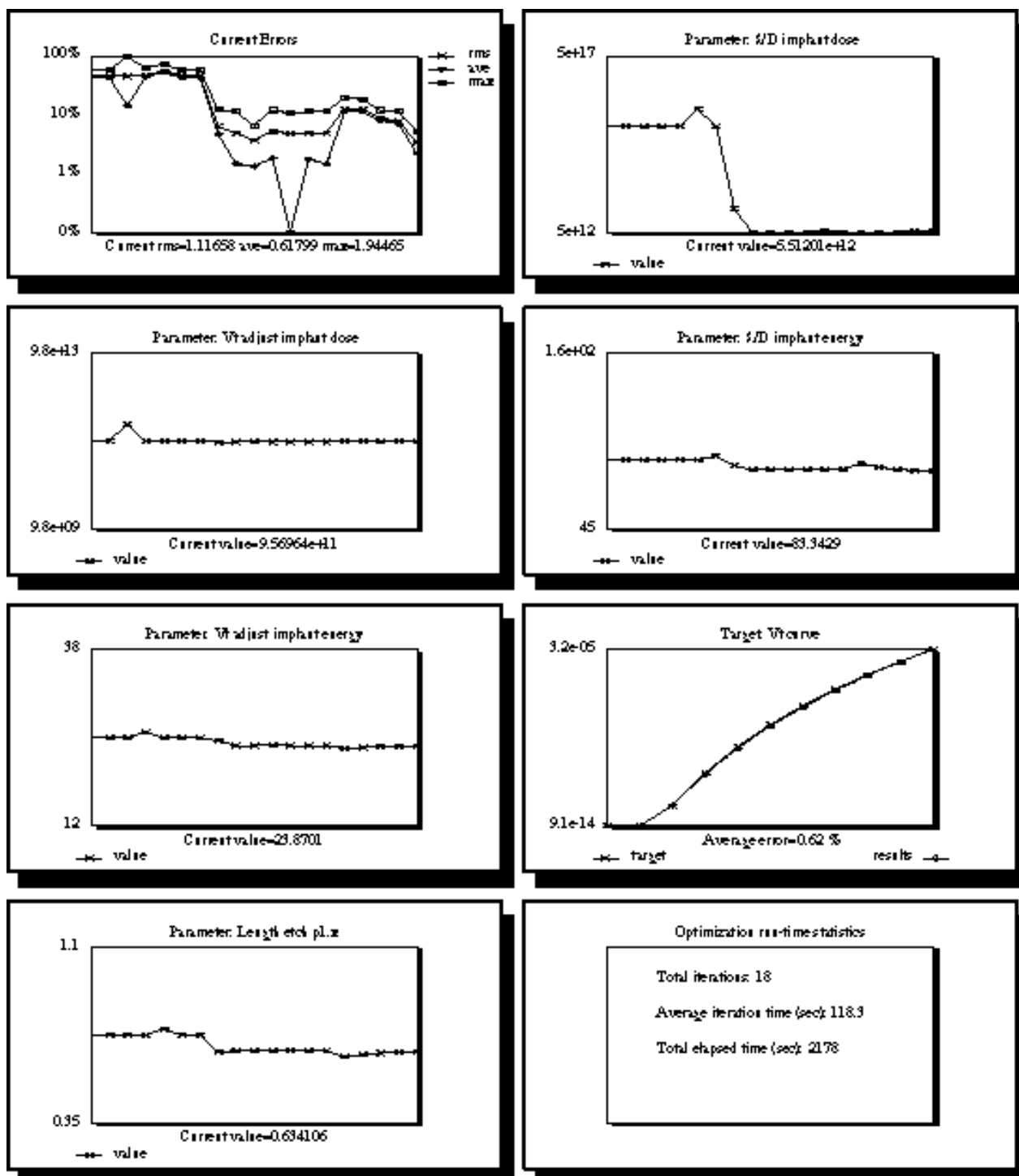


Figure 6-31: Graphic Printout

6.11: Optimization Tuning

This section contains information useful when an optimization run does not converge for some reason. It indicates some common errors and shows how different setup, parameter, and target values are designed to work.

Possible error conditions include the following:

- **Input parameters have little effect on extracted results:** This is usually indicated by very little target movement (as seen on the **Graphics** display) during the optimization. Use different parameters, increase the parameter min/max range, or change the parameter type from **linear** to **log**.
- **Targets conflict with each other:** This is an extremely common problem. For instance, asking the OPTIMIZER to optimize to a thick gate oxide thickness and to a low threshold voltage (that isn't feasible with the thick oxide) is doomed to fail. The best way to get around these types of problems is by carefully choosing targets and target values, plus changing the weighting of some targets to make them more (or less) sensitive than others.

You can make a target less sensitive by setting its weight to a number between 0 and 1. Numbers greater than 1 make it more sensitive (i.e., it must be met with a smaller percentage error).

- **Input parameters hit their minimum or maximum ranges:** This can happen if the initial value is not very close to the value required to achieve optimization. Increase the min/max range of these parameters
- **Failure to converge:** If all the other error conditions are taken care of and the optimization still does not converge, it's often because just a few target points can not be met, or the OPTIMIZER's initial values were too far off the mark.

First, check the error percentages on each target. If most were close but just a few have large error terms, then you might consider decreasing the weight of those targets or changing them from **linear** to **log** type. If those points are part of a curved target, you might consider deleting some or all of them.

On the other hand if the OPTIMIZER fails, try better initial values, decrease the parameter min/max ranges or disable the parameters that have the strongest effect on the results.

Another common reason for convergence failure is a bad curved target. Keep curved targets smooth, without excessive points, and if necessary cut down the min/max range of especially dominant input parameters.

You can also try increasing the number of iterations, but the default value of 4 should almost always be adequate.

Note: As an aid to optimization, the OPTIMIZER uses Optimized value as each input parameter's initial value after the OPTIMIZER has been run once. To use the Initial value once again, choose **Reset Values** on the **Parameter Edit** menu. The optimized values are deleted on the Parameter worksheet.

A.1: Introduction

Models and Algorithms used by one dimensional (1D) electrical solvers in DECKBUILD and TONYPLOT.

Note: This appendix is intended to serve as a quick reference only. A detailed description of the semiconductor device physical models is provided in the ATLAS manual.

1D electrical solvers, available by using the `extract` command in DECKBUILD or in TONYPLOT, are based on the iterative solution of the Poisson equation:

$$\text{div}(\varepsilon \nabla \psi) = q(p - n + N_D^+ - N_A^-) - \rho_F \quad \text{A-1}$$

where ψ is the potential, ε is the dielectrical permittivity, n and p are the electron and hole concentrations, and ρ_F is the fixed charge.

QUICKBIP uses the continuity equations to calculate n and p :

$$\frac{1}{q} \text{div} J_n^{\otimes} - U_n^{\otimes} = 0 \quad \text{A-2}$$

$$\frac{1}{q} \text{div} J_p^{\otimes} - U_p^{\otimes} = 0 \quad \text{A-3}$$

where:

$$J_n^{\otimes} = q \mu_n E_n^{\otimes} \cdot n + q D_n \nabla n \quad \text{A-4}$$

$$J_p^{\otimes} = q \mu_p E_p^{\otimes} \cdot p + q D_p \nabla p \quad \text{A-5}$$

$$D_n = \frac{kT}{q} \mu_n, \quad D_p = \frac{kT}{q} \mu_p \quad \text{A-6}$$

A.1.1: Physical Models

All electrical solvers take into account the following models and effects:

- Temperature dependence, such as kT/q or Eg
- Concentration-dependent mobility (with built-in temperature dependence)
- Field-dependent mobility (perpendicular field with built-in temperature dependence)
- Material work function (for MOS structures)
- Fixed interface charge

A.2: Concentration Dependent Mobility

The concentration dependent mobilities for n and p respectively are:

$$\mu_n^D = \mu_{nmin} + \frac{\Delta\mu_n}{1 + N_{total}/N_{nref}} \quad A-7$$

$$\mu_p^D = \mu_{pmin} + \frac{\Delta\mu_p}{\Delta + N_{total}/N_{pref}} \quad A-8$$

where:

$$\mu_{nmin} = 88 \cdot \left(\frac{Y}{300}\right)^{-0.57} \quad A-9$$

$$\mu_{pmin} = 54.3 \cdot \left(\frac{Y}{300}\right)^{-0.57} \quad A-10$$

$$\Delta\mu_n = 1252 \cdot \left(\frac{Y}{300}\right)^{-2.33} \quad A-11$$

$$\Delta\mu_p = 407 \cdot \left(\frac{Y}{300}\right)^{-2.33} \quad A-12$$

$$N_{nref} = 1.432 \cdot 10^{17} \left(\frac{Y}{300}\right)^{2.456} \quad A-13$$

$$N_{pref} = 2.67 \cdot 10^{17} \left(\frac{Y}{300}\right)^{2.456} \quad A-14$$

A.3: Field Dependent Mobility Model

The field dependent mobilities for n and p respectively are:

$$\mu_n = \frac{\mu_n^D}{\sqrt{1 + 1.54 \cdot 10^{-5} \cdot E}} \quad \text{A-15}$$

$$\mu_p = \frac{\mu_p^D}{\sqrt{1 + 5.35 \cdot 10^{-5} \cdot E}} \quad \text{A-16}$$

A.4: Sheet Resistance Calculation

After solving the Poisson equation, the sheet resistance for each semiconductor layer is estimated using:

$$R_{sh} = \int_{xleft}^{xright} \frac{dx}{q\mu_n \cdot n + q\mu_p \cdot p} \quad \text{A-17}$$

$xleft$ and $xright$ are determined by the p - n junction locations and the semiconductor material boundaries.

A.5: Threshold Voltage Calculation

Threshold voltage calculation is based on the calculated sheet resistance. In MOS mode (1-D vt extraction), the solver will calculate threshold voltage automatically. First, the conductance of the channel region will be calculated for each gate voltage applied. If an NMOSFET structure is assumed, then:

$$g(V_g) = \int_O^{x_{inv}} (q\mu_n)/n dx \quad A-18$$

O corresponds to the oxide-silicon interface and x_{inv} is the boundary of the inversion layer. Threshold voltage will be determined using the $g(V_g)$ curve as an intersection with the V_g axis of the straight line drawn through two points on the $g(V_g)$ curve, corresponding to the maximum slope region shown below.

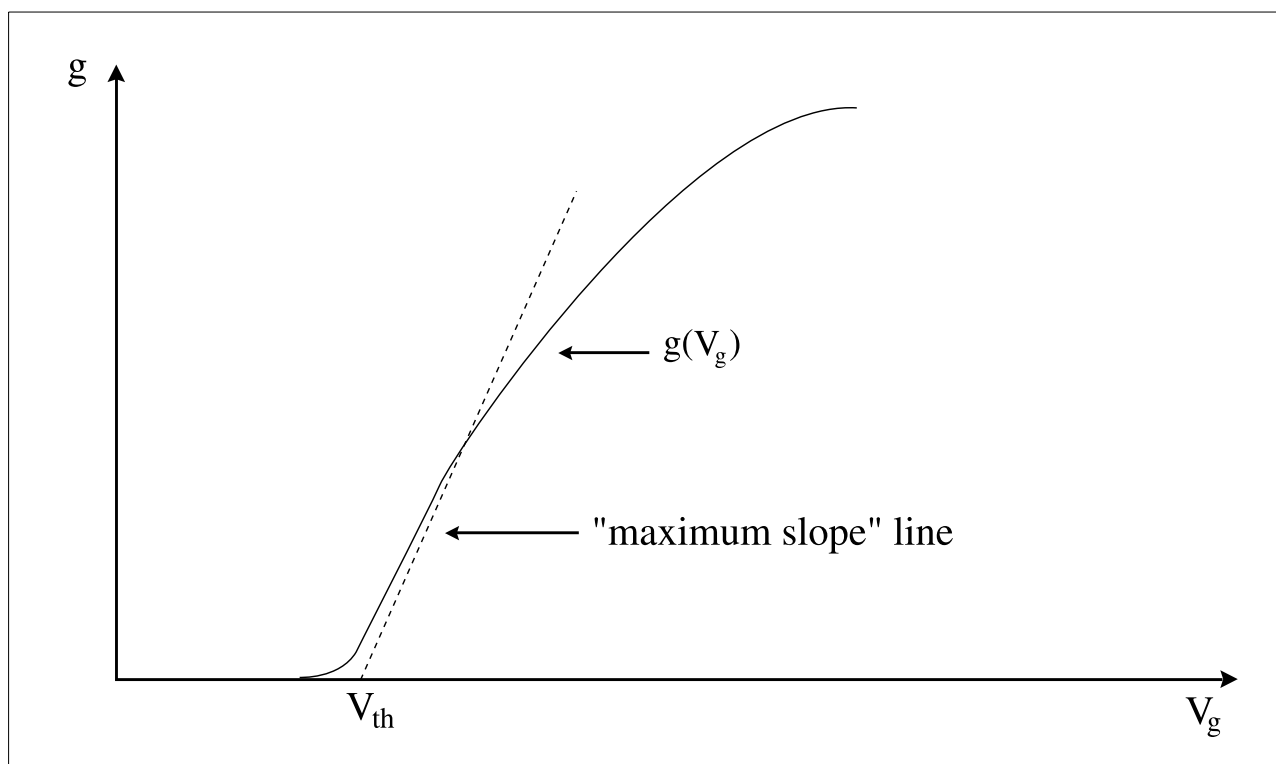


Figure A-1: Threshold Voltage Calculation

A.5.1: Breakdown Voltage Calculation

Breakdown voltage calculation is based on estimation of ionization integrals for electrons and holes. Breakdown is determined by the condition that one of the integrals is greater than 1. The ionization rates are calculated using the following equations (See the Selberherr model in the ATLAS manual):

$$\alpha_n = AN \cdot \exp\left[-\left(\frac{BN}{E}\right)^{\text{BETAN}}\right] \quad \text{A-19}$$

$$\alpha_p = AP \cdot \exp\left[-\left(\frac{BP}{E}\right)^{\text{BETAP}}\right] \quad \text{A-20}$$

where:

AN = AN1 if E < EGRANAN = AN2 if E > EGRAN
AP = AP1 if E < EGRANAP = AP2 if E > EGRAN
BN = BN1 if E < EGRANBN = BN2 if E > EGRAN
BP = BP1 if E < EGRANBP = BP2 if E > EGRAN

The values of the parameters AN1, AN2, AP1, AP2, BN1, BN2, BP1, BP2, BETAN, BETAP, EGRAN are user-definable (through the `extract` command or pop-up menu). Their default values are:

AN1=7.03e5 cm⁻¹
AN2=7.03e5 cm⁻¹
BN1=1.231e6 V/cm
BN2=1.231e6 V/cm
AP1=6.71e5 cm⁻¹
AP2=1.582e6 cm⁻¹
BP1=1.693e6 V/cm
BP2=2.036e6 V/cm
BETAN=1.0 (unitless)
BETAP=1.0 (unitless)
EGRAN=4e5 V/cm

B.1: DBInternal

DBINTERNAL is a simple but powerful DECKBUILD tool that allows you to create a Design Of Experiments (DOE) from a pair of input files. Amongst other things, you can create corner models for process parameters or device characteristics or both.

Any parameters that are to be used as variables must be specified as `set` statements in a template file. Any results of interest should be calculated using `extract` statements.

The DOE is specified with simple `sweep` statements in a separate design file. The `sweep` statement defines which variables are required in the DOE, and the range of values these variables are to take.

The parameter values and the results of each simulation can be stored in a file that can be viewed in TONYPLOT or used as a data base for input to a statistical analysis tool such as SPAYN.

B.1.1: Example

Suppose you have an ATLAS deck (`resistor_template.in`) for a simple resistor (the doping is controlled by a `set` statement). When run, this deck calculates the resistance from the gradient of the VI curve.

```
go atlas
set doping=1e16
mesh width=2
x.mesh loc=0.0 spac=0.25
x.mesh loc=1.0 spac=0.25
y.mesh loc=0.0 spac=0.25
y.mesh loc=10.0 spac=0.25
region num=1 silicon
electrode num=1 top name=ground
electrode num=2 bottom name=anode
doping uniform n.type conc=$doping
models conmob
solve init
log outf=dop$doping'.log
solve vanode=0.0 vstep=0.1 vfinal=2.0 name=anode
log off
extract init infile="dop$doping'.log"
extract name="res" grad from curve(i."anode",v."anode") where y.val=1
quit
```

If you want to investigate how doping affects the resistance, you can create a DBINTERNAL deck (sweep.in) that defines an experiment (a series of trials). In this example, the doping is changing between 10^{15} and 10^{19} cm^{-3} (at three points per decade, thirteen points in all).

```
go internal
load infile=resistor_template.in
sweep parameter=doping type=power range="1.0e15, 1.0e19, 13"
save type=sdb outfile=resistance.dat
quit
```

When you execute sweep.in, DBINTERNAL runs the resistor_template.in deck several times, each time changing the value on the set doping= line. DBINTERNAL also collates the data generated by the extract name="res" line and saves it in sdb format suitable for viewing with TONYPLOT. The resistance as a function of doping is shown in the following figure.

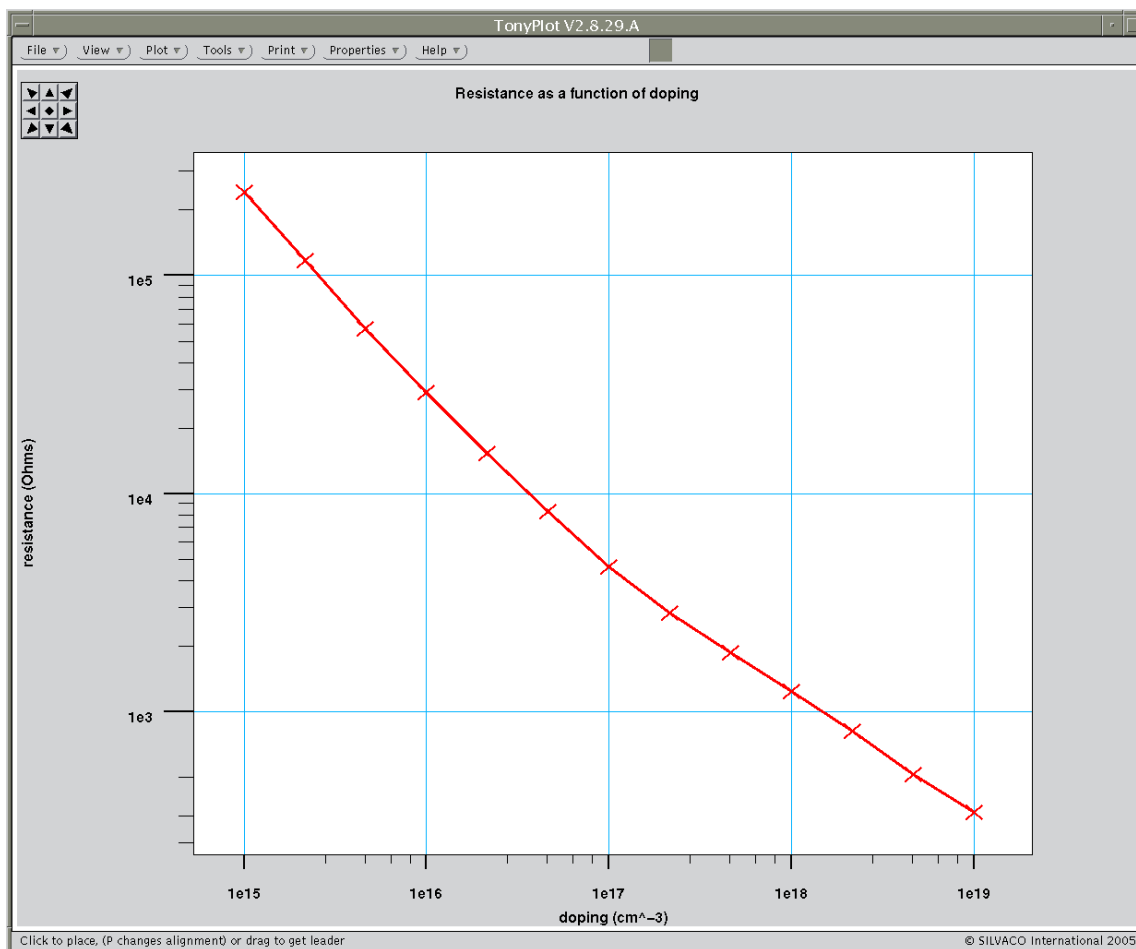


Figure B-1: Doping Resistance

B.2: The Template File

The template file is a description of the class of simulations you want to perform. It should be a deck that will execute correctly when run within DECKBUILD.

Any variables that you need DBINTERNAL to control must be defined on a `set` line. For example, the file `resistor_template.in` has the line

```
set doping=1e16
```

so DBINTERNAL can change the value of the variable `doping`. DBINTERNAL ignores the actual value on the `set` line in the template file. It is safe to set variables that DBINTERNAL doesn't control. They will remain with the value defined in the template file.

The variable is normally used to set numbers in the template file. For example:

```
doping uniform n.type conc=$doping
```

where the doping concentration is being set by the variable `doping`. But it is also useful to be able to use the value as a string in a filename. In this instance, you should enclose the variable name in single quotes. For example:

```
log outf=dop$doping'.log
extract init infile="dop$doping'.log"
```

So if `doping` had been set to `1e16`, the filename would be `"dop1e16.log"`.

The template file may have `extract` statements. For example:

```
extract init infile="dop$doping'.log"
extract name="res" grad from curve(i."anode",v."anode") where y.val=1
```

DBINTERNAL will recognize you are interested in the result `"res"` (for example) and will collect these results from each simulation.

B.3: The Experiment File

The experiment file has three main parts

B.3.1: Load command

```
load infile=resistor_template.in
```

This tells DBINTERNAL which file to use as the basis for the simulations.

B.3.2: Experiment command

```
sweep parameter=doping type=power range="1.0e15, 1.0e19, 13"
```

This tells DBINTERNAL how you want the variables to change.

B.3.3: Save Command

If the template file contains extract statements, you also want a save command

```
save type=sdb outfile=resistance.dat
```

which tells DBINTERNAL where to save the extracted data. The saved file will contain the values of all the independent variables (the variables defined in the experiment command) and the values of all the dependent variables (the variables calculated with extract statements).

B.4: Technical Details

DBINTERNAL reads in the template file and looks for any variables defined on an `extract` line and makes a note of their names. The name must be the first parameter after the `extract` command and have no spaces.

```
extract name="res" ...
```

DBINTERNAL also knows what parameters have been set on the `experiment` line. For example, DBInternal will control the parameters `x` and `y`.

```
sweep parameter=x type=linear range=1,2,2 \
      parameter=y type=linear range=3,5,3
```

To run a trial, DBINTERNAL creates a temporary `<infile>` with the name

```
<infile>=dbinternal_temporary_<name>_<pid>
```

`<name>` is the name of the machine and `<pid>` is the program ID of the DBINTERNAL program. (See also the “`log`” command). The temporary file is a copy of the template file with different values on any set line that correspond to a parameter in the `experiment` command. For instance, if the template file had the lines

```
set x=5
set y=10
set z=15
```

and you ran the earlier `sweep` command the first temporary file would have the lines

```
set x=1
set y=3
set z=15
```

DBINTERNAL will change the values for parameters it recognizes and leaves the other set lines alone.

DBINTERNAL then runs a trial by producing a child (DECKBUILD) with the command

```
deckbuild [-v a.b.c.X][-int][-ascii][-noplot] -run <infile> -outfile
<infile>.out
```

Once DECKBUILD is finished, DBINTERNAL will parse the `<infile>.out` file to find the values generated by the `extract` statements. The `<infile>` and the `<infile>.out` are then deleted.

This procedure (creating an `<infile>`, starting DECKBUILD, examining the `<infile>.out`) is repeated for the remaining sets of parameters in the experiment.

B.5: DBInternal Commands

DBINTERNAL commands are generally of the form

```
<command> <param1>=<value1> <param2>=<value2> ...
```

The commands and the parameters may be abbreviated but they must be long enough to be recognized. For instance, the save command may be shortened to sa, but not to s because that would not distinguish between save and sweep.

If the value contains whitespace, it must be enclosed in quotes. For instance

```
range="1.0e15, 1.0e19, 13"
```

But if the value is a single block of text, there is no need for the quotes. This range can be entered as

```
range=1.0e15,1.0e19,13
```

DBINTERNAL recognizes the following commands.

B.5.1: doe

Syntax

```
doe type=<doe_type> \  
    parameter=<param1> range="center, delta" \  
    parameter=<param2> range="center, delta"
```

Description

The trials of a DOE experiment correspond to various points on or near a hypercube around some origin in parameter space. The results can be used to create a model for the dependent variables over the hypercube.

For example, suppose you had a process that generated FETs with gate lengths in the range 95-105 μm and recess depths in the range 45-55 μm . The parameter space is a square with corners (95, 45), (95, 55), (105, 45) and (105, 55). Suppose you want to know how breakdown voltage is affected by these parameters. If you knew there were a simple linear relationship, you could simulate at the midpoint and the two points where the axis intersected with the hypercube ((100, 50), (105, 50), (100, 55)) and fit a simple linear model through the results. If you thought there were a more complex relationship, you would simulate at more points over the hypercube. For example, the midpoint and all the corners.

A parameter should be the name of a variable in the template deck. The names (e.g., <param1>) cannot be abbreviated. They must be exactly as they appear in the template deck.

The range is two numbers. The first is center of the hypercube (i.e., the value of the parameter at the middle of its range). The second is the distance from the center to the edge of the hypercube (or half the range of the parameter).

The points are almost always high symmetry points on the hypercube, such as the corners of the hypercube, the points where an axis intersects the hypercube, and a midpoint along an edge of the hypercube. The best way to see the points generated by a DOE type is to use the no_exec command and setting the range of the parameters to "0, 1". Therefore, 0 indicates a center point, 1 indicates one side of the hypercube and -1 the other side.

Example

```
no_exec outfile=tlff.dat
doe type=two_level_full_factorial \
    parameter=p1 range=0,1 \
    parameter=p2 range=0,1 \
    parameter=p3 range=0,1
```

DOE Types

The DOE type must one of the following:

- `gradient_analysis`
- `two_level_full_factorial`
- `two_level_half_factorial`
- `three_level_full_factorial`
- `face_centered_cubic`
- `circumscribed_circle`
- `box_behnken`

gradient_analysis

This type does a simulation at the center point and at the points one positive step along each axis. An experiment with N parameters has $N+1$ trials.

two_level_full_factorial

This type does a simulation corresponding to every node of the N -dimensional hypercube. An experiment with N parameters has 2^N trials.

two_level_half_factorial

This type does half the simulations of the `two_level_full_factorial` and no two nodes are on the same edge. An experiment with N parameters has 2^{N-1} trials.

three_level_full_factorial

This type does a simulation corresponding to every node and every half point of the N -dimensional hypercube. An experiment with N parameters has 3^N trials.

face_centered_cubic

This type does a simulation corresponding to every node. Every point where an axis intersects the hypercube and the center point. An experiment with N parameters has $2^N + 2N + 1$ trials.

circumscribed_circle

This type is similar to the `face_centered_cubic` type but the axis points now lie on the surface of the hypersphere that passes through the hypercube node points (i.e., all points are equidistant from the origin). An experiment with N parameters has $2^N + 2N + 1$ trials.

box_behnken

The design matrix for this simulation is based on a balanced or partially balanced block design. There is no easy relationship between the number of parameters and the number of trials in the experiment. For example, an experiment with seven parameters requires 57 trials and an experiment with nine parameters requires 97 trials. But an experiment with eight parameters is particularly inefficient and requires 225 trials.

B.5.2: endsave

Syntax

```
endsave
```

Description

This command tells DBInternal to stop saving data to the current file. See the `save` command for more information.

Example

```
save type=sdb outfile=example.out
sweep parameter=doping type=power range=1e15,1e19,13
endsave
```

This stops DBINTERNAL from trying to save any more data to `example.out`.

B.5.3: log

Syntax

```
log outfile=<filename_root>
```

Description

This command tells DBINTERNAL to keep the output generated by the child DECKBUILD for each trial. When generating the temporary input file for a trial, DBINTERNAL uses the `<filename_root>` and appends the ID of the trial (for example, the first trial has an ID of zero and the second trial has an ID of one). The usual command is issued to run the trial

```
deckbuild -int -noplot -run <infile> -outfile <infile>.out
```

At the end of the trial, only the `<infile>` is deleted and the `<infile>.out` will remain.

Example

```
load infile=example.in
log outfile=keep
sweep parameter=doping type=linear range=1,4,4
```

This will generate the files `keep1.out`, `keep2.out`, `keep3.out`, and `keep4.out`.

B.5.4: monte_carlo

Syntax

```
monte_carlo number=<num_trials> \
    parameter=<param1> type=<mc_type> coeffs="list, of, numbers" \
    parameter=<param2> type=<mc_type> coeffs="list, of, numbers"
```

Description

The `monte_carlo` command generates an experiment from a specified number of trials. All parameter values in each trial are random. Each parameter is drawn from its own distribution.

The number is the number of trials you want in the experiment.

A parameter should be the name of a variable in the template deck. The names (e.g., `<param1>`) cannot be abbreviated. They must be exactly as they appear in the template deck.

The type must be one of the following:

- uniform
- normal
- log_normal
- gamma
- weibull

These are the types of random distributions a parameter will be extracted from. The coeffs are a list of numbers that describe the random distribution, the number and meaning of the coeffs depending upon which distribution was chosen.

Random Distributions

The probability density function and the required coefficients for the following random distributions.

uniform

The uniform type takes random numbers evenly spaced between two limits. The probability density function is

$$p(x) = 0 \quad (x < x_{lo} \text{ or } x \geq x_{hi})$$

$$p(x) = \frac{1}{x_{hi} - x_{lo}} \quad (x_{lo} \leq x < x_{hi}) \quad \text{B-1}$$

This distribution needs two coefficients, the minimum and maximum allowed values

$$coeffs = "x_{lo}, x_{hi}" \quad \text{B-2}$$

normal

The normal type is a Gaussian probability density. The probability density function is

$$p(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \quad \text{B-3}$$

This distribution needs two coefficients, the mean and the standard deviation.

$$coeffs = "\mu, \sigma" \quad \text{B-4}$$

log_normal

The log_normal type has a probability density function of (the distribution of log(x) would be normal)

$$p(x) = \frac{1}{\sqrt{2\pi} \cdot X\sigma} \exp\left(-\frac{(\log(X) - \mu)^2}{2\sigma^2}\right) \quad \text{with} \quad X = \pm(x - \tau) \quad \text{B-5}$$

This distribution needs four coefficients (the last coefficient should be +1 or -1 and gives the sign in front of X).

$$coeffs = "a \mu, \sigma, \pm" \quad \text{B-6}$$

gamma

The `gamma` type is the time to wait for several events that occur with a Poisson distribution. The probability density function is

$$p(x) = X^{a-1} \cdot \left(-\frac{\exp(X)}{\sigma \cdot \Gamma(a)} \right) \quad \text{with} \quad X = \pm \left(\frac{x-\mu}{\sigma} \right). \quad \text{B-7}$$

This distribution needs four coefficients (the first coefficient should be a +ve integer and the last should be +1 or -1).

`coeffs = "a μ, σ, ± "` B-8

Weibull

The probability density function of the `weibull` type is

$$p(x) = \frac{a}{\sigma} \cdot X^{a-1} \cdot \exp(-X^a) \quad \text{with} \quad X = \pm \left(\frac{x-\mu}{\sigma} \right) \quad \text{B-9}$$

This distribution needs four coefficients (the last coefficient should be +1 or -1).

`coeffs = "a μ, σ, ± "` B-10

B.5.5: no_exec**Syntax**

```
no_exec type=<ssf|spayn> outfile=<filename>
```

Description

This is a debugging command. If it is issued before an experiment, then DBINTERNAL will generate a file `<filename>`. This file contains a list of the independent variables, which would have been passed to the simulations. No actual trials will be performed. An `ssf` format file is a simple list of numbers suitable for viewing in a text editor. It can also be plotted in TONYPLOT. A `spayn` format file can be analyzed with SPAYN.

Example

If you run an experiment file

```
no_exec type=ssf outfile=example.out
sweep parameter=length type=linear range=1,2,2 \
parameter=width type=linear range=1,3,3
```

the data in `example.out` would be

```
...
0 1 1
1 2 1
2 1 2
3 2 2
4 1 3
5 2 3
```

The first column is the ordinal of the trial. The second column is the length value of that trial (e.g., 1 or 2). The third column is the width value of that trial (e.g., 1, 2 or 3).

B.5.6: option

Syntax

```
option [[!]int][[!]ascii] [[!]plot][version=<string>]
```

Description

The `option` command controls some of the command line options passed to DECKBUILD when running a trial.

The value of a parameter (`int`, `ascii`, or `plot`) is either true or false. If the parameter is present, it is set to true. If it is negated (with the `!`), it is set to false. If it is absent, it is set to a default value. The default for `int` is true. The default for both `ascii` and `plot` is false.

The `version` parameter defines which version of DECKBUILD to use to run the child process.

A trial is run with the command

```
deckbuild [-v a.b.c.X][-int][-ascii][-noplot] -run <infile> -outfile  
<infile>.out
```

The `-int` option tells DECKBUILD to start DBINTERNAL as the default simulator. The `-ascii` option tells DECKBUILD not to start its GUI. The `-noplot` option tells DECKBUILD to ignore any TONYPLOT commands in `<infile>`.

The `-int` option will be passed if `int` is true and will be absent if `int` is false.

The `-ascii` option will be passed if `ascii` is true and will be absent if `ascii` is false.

The `-noplot` option will be passed if `plot` is false and will be absent if `plot` is true.

Example

```
option ascii !plot
```

This will run the trials without the DECKBUILD GUI and ignoring any TONYPLOT commands in the trial deck.

```
option !ascii plot version=3.22.4.C
```

This will run the trials, using version 3.22.4.C of DECKBUILD, inside a child DECKBUILD GUI and will execute any TONYPLOT commands in the trial deck.

B.5.7: save

Syntax

```
save type=<sdb|spayn> outfile=<filename>
```

Description

The `save` command saves the data generated by the experiment in the file `<filename>`. You can output the data in `sdb` format (to be viewed in TONYPLOT) or in `spayn` format (to be analyzed by SPAYN).

The following data is stored for each trial:

- The ID of the trial.
- The values of the parameters defined on the experiment line.
- The values of the parameters calculated with `extract` commands.

You can place the `save` command before or after the experiment line. If it comes before the experiment line, the file will be rewritten at the end of each trial. Therefore if something unforeseen happens during an experiment, you will have the data from the trials that were completed. If it comes after the experiment line, all the data from the experiment will be written at once.

Only one file at a time can be active. If you define two `save` statements before an experiment line, only the second will actually get the following data.

```
save type=sdb outfile=save.sdb

save type=spayn outfile=save.spayn

sweep parameter=doping type=power range=1e15,1e19,13
```

The file `save.sdb` will have no data (the file `save.spayn` will be fine). Once a file is active, it will remain active until a subsequent `save` command makes a different file active or until an `endsave` command is given. A file, however, will only have data from one experiment.

Warning: If you have more than one experiment line in a deck, be very careful with the `save` command or you will lose data. For example, the following is the wrong way.

```
sweep parameter=doping type=power range=1e15,1e19,13

save type=sdb outfile=one.sdb

sweep parameter=doping type=linear range=1e16,1e17,11

save type=sdb outfile=two.sdb
```

This will perform the first experiment, then save that experiment to `one.sdb`, then perform the second experiment. But because `one.sdb` is still the active file, `DBINTERNAL` will write the data from the second experiment to `one.sdb`, destroying the data that was already there. At the end of this run, both `one.sdb` and `two.sdb` will contain the same data. In this case, you must use the `endsave` command to tell `DBINTERNAL` to deactivate the active file.

Example

```
sweep parameter=doping type=power range=1e15,1e19,13

save type=sdb outfile=one.sdb

endsave

sweep parameter=doping type=linear range=1e16,1e17,11

save type=sdb outfile=two.sdb
```

B.5.8: sweep

Syntax

```
sweep parameter=<param1> type=<sweep_type> range="start, stop, num" \
      parameter=<param2> type=<sweep_type> data="point ... list"
```

Description

The `sweep` command generates an experiment from all combinations of individual parameter values. The first parameter changes with the highest frequency. The final parameter changes with the lowest frequency.

A parameter should be the name of a variable in the template deck. The names (e.g., `<param1>`) cannot be abbreviated. They must be exactly as they appear in the template deck.

The type must be either linear, power or list.

The range is three numbers, the initial value of the parameter, the final value of the parameter, and the number of points. This is used to for the linear and power types.

Example 1

In a linear sweep, the parameter values are evenly spaced.

```
sweep parameter=x type=linear range="1,4,7"
```

This generates for x the values:

- 1
- 1.5
- 2
- 2.5
- 3
- 3.5
- 4

Example 2

In a power sweep, the log of the parameter values are evenly spaced.

```
sweep parameter=y type=power range="1e10, 1e15, 6"
```

This generates for y the values:

- 1e10
- 1e11
- 1e12
- 1e13
- 1e14
- 1e15

Example 3

The data is a list of values to assign to the parameter.

```
sweep parameter=z type=list data="1,2.5,3,3.1,4"
```

This assigns each of the values (one at a time) to z. The number of trials in the experiment is the product of the number of points for each parameter.

Example 4

```
sweep parameter=x type=linear range="1,4,7" \  
parameter=y type=power range="1e10, 1e15, 6"
```

This generates an experiment with 42 trials. All the values of x in combination with all the values of y. Adding another variable:

```
sweep parameter=x type=linear range="1,4,7" \  
parameter=y type=power range="1e10, 1e15, 6" \  
parameter=z type=list data="0, 2, 3"
```

would increase the number of trials to 126. All 42 trials from the previous experiment with z=0, and all 42 trials with z=2, and all 42 trials with z=3.

B.6: DBIT

DBIT is a GUI tool that provides some of the functionality of DBINTERNAL. DBIT runs outside of DECKBUILD. But DECKBUILD is still required to run the child trials. To start DBIT, type "dbit" at the command line. The DBIT Main Window will then appear.

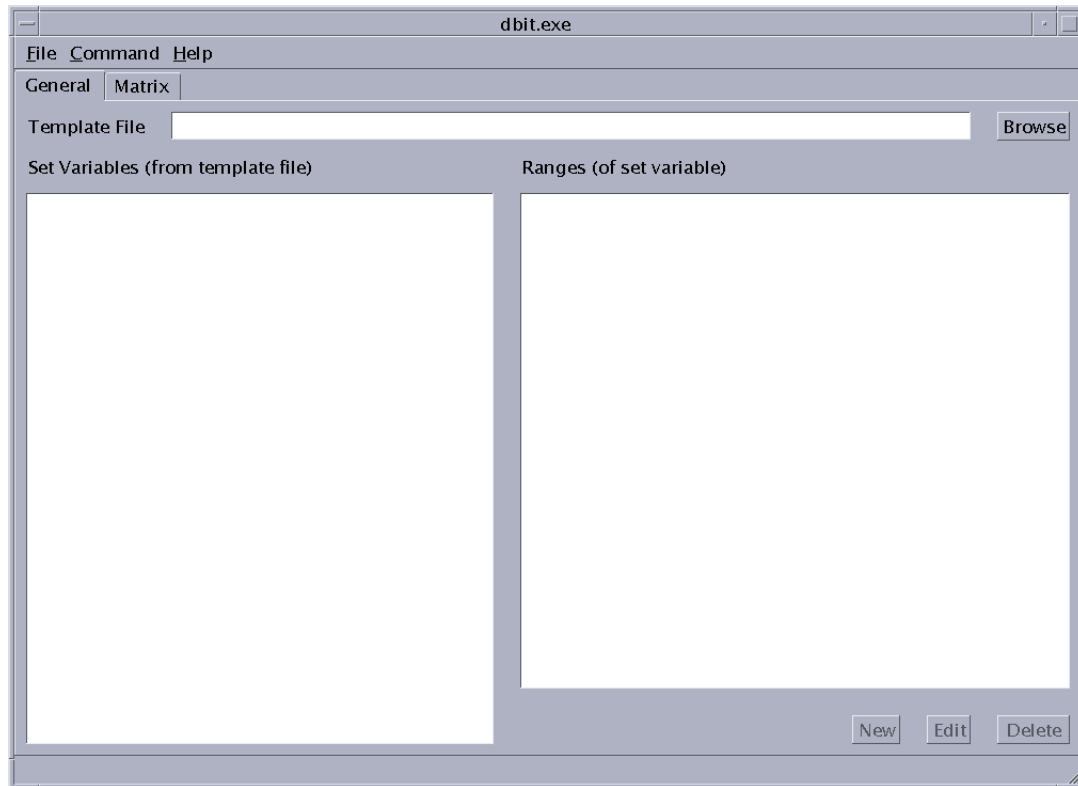


Figure B-2: DBIT Main Window

B.6.1: The General Tab

Click on the **Browse** button or select **File→Open** to open a template file. This file will be analyzed and the variables that appear on the "set" lines of the template file will be listed in the left hand box (see Figure B-3).

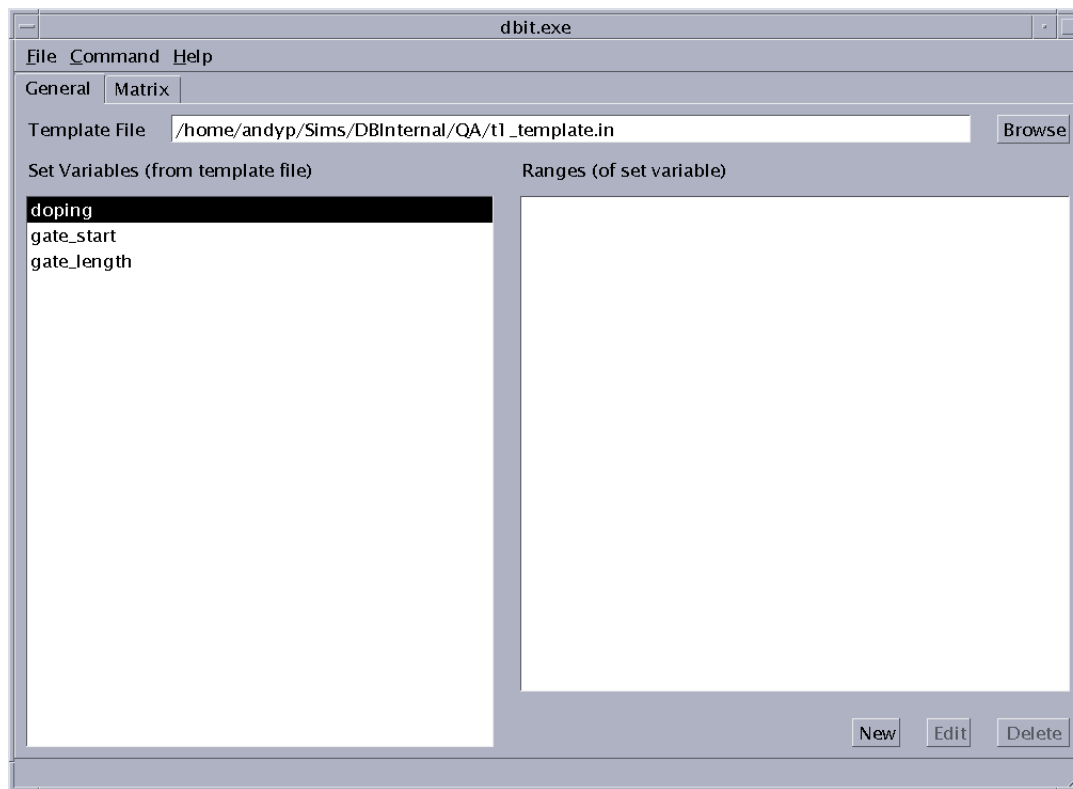


Figure B-3: General Tab

Note: The input file must be a template file rather than a dbinternal experiment file.

To define a range of values for a variable, highlight the appropriate variable in the left hand box and click the **New** button or double click the name of the variable. The Range Dialog Window will appear (see Figure B-4).

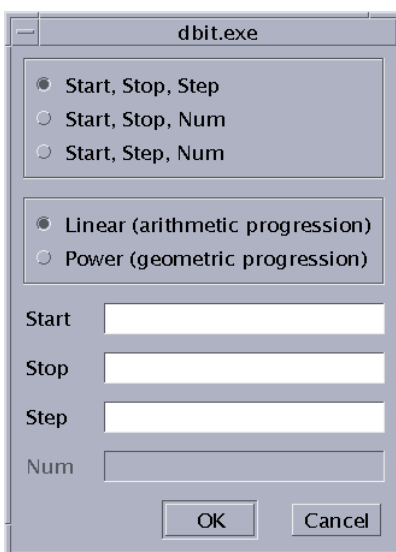


Figure B-4: Range Dialog Window

The range can be defined as an arithmetic progression (Equation B-11) or a geometric progression (Equation B-12).

$$a_1, a_1 + d, a_1 + 2 \cdot d, a_1 + 3 \cdot d, \dots a_n = a_1 + (n - 1) \cdot d \quad \text{B-11}$$

$$a_1, a_1 \cdot r, a_1 \cdot r^2, a_1 \cdot r^3, \dots a_n = a_1 \cdot r^{(n-1)} \quad \text{B-12}$$

There are four parameters associated with a range:

- the initial value (a_1)
- the number of points (n)
- the step (d for arithmetic progression and r for geometric progression)
- the final value (a_n).

A range can, in principal, be uniquely defined by giving any three of these parameters. But DBIT expects you always to define the initial value, so there are three ways to define a range:

- the initial value, the final value, and the step
- the initial value, the final value, and the number of points
- the initial value, the step, and the number of points

If you require a single value just enter it into Start.

If you use Start, Stop, Step to define the range, the final value of the range may not be the value explicitly written in Stop. If the progression defined by Start and Step does not exactly reach the Stop value, then the range will stop before the Stop value. For example, Start=1, Stop=10, Step=2 will result in the range "1,3,5,7,9" and not "1,3,5,7,9,10".

Once you add a range, you can edit the description of the range by highlighting the range (in the right hand box) and clicking the **Edit** button (or by double clicking on the appropriate range). You can delete a range by highlighting it and clicking the **Delete** button.

B.6.2: The Matrix Tab

Once you finished defining the ranges for the variables, click on the **Matrix** tab. This shows a spreadsheet-like area that lists all the trials to be done (see Figure B-5).

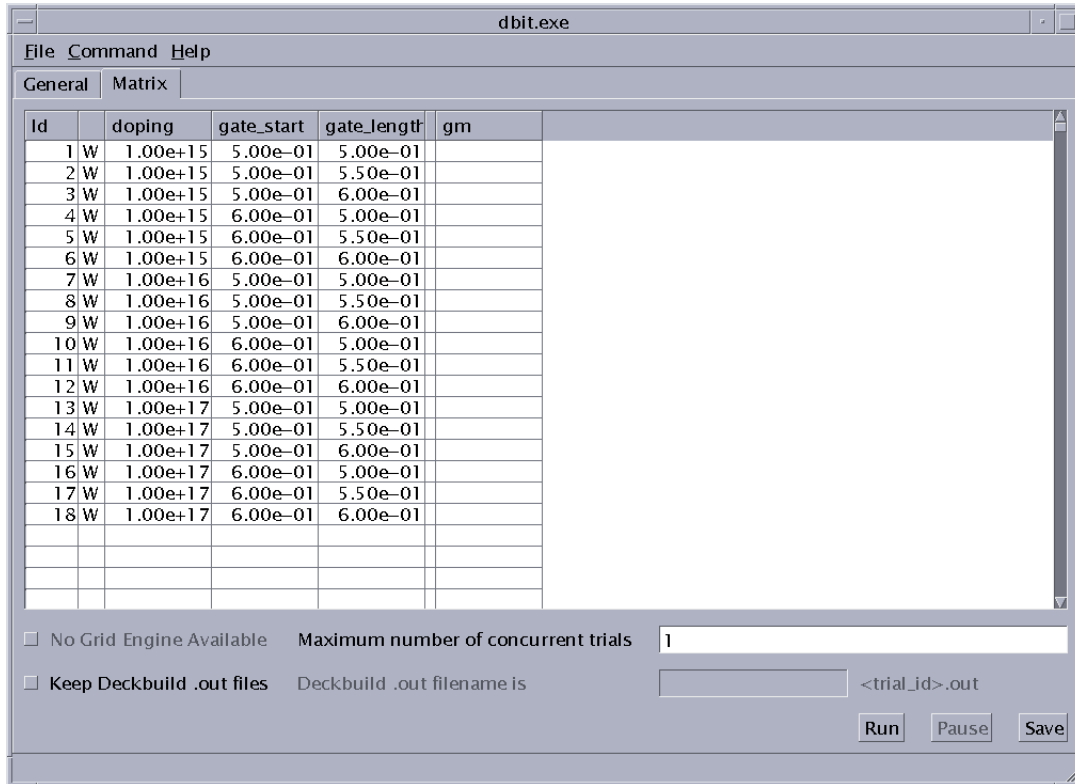


Figure B-5: Matrix Tab

Click the **Run** button to run the trials. The trials are run in the order they are defined in the spreadsheet. When one trial is finished, a new trial starts with the next available set of numbers.

If you click the **Pause** button, no new trials will be started (until you click on the **Run** button again) but any currently running trials will be allowed to finish.

Each row in the spreadsheet corresponds to a trial. You can edit various cells in the spreadsheet to change or add trials to the experiment. Clicking the right mouse button while the cursor is over the spreadsheet brings up a menu with the available commands.

The first column is **Id**. This is an integer that is assigned by DBIT to a trial. Each row will have a unique integer. If you add a trial to the list, it will automatically assign the next available Id. You cannot change the value in this column. The Id assigned to a trial is available within the template file by using \$trial_id. For example, you can output data in the template file data with the command

```
save outfile=data_ '$trial_id'.dat
```

This will save the data to a different file for each trial.

The second column is a single character that gives the current status of the trial. A 'W' means the trial is waiting to run. An 'R' means the trial is currently running (or has finished running but the output files have not yet been processed). An 'F' means the trial finished successfully. An 'X' means the trial failed to start. An 'H' means you have hidden the trial.

A hidden trial is still displayed in the spreadsheet but DBIT will not run it. To hide a trial, select the rows that correspond to the trials you want to hide, click the right mouse button, and select **Hide Trial**. You can reactivate a hidden trial by selecting the **Activate Trial** option.

The next few columns in the spreadsheet correspond to the set variables, one column for each variable (the name of the variable is at the top of the column). The values in the columns are the values that will be assigned to the variables when the corresponding trial is running. You can edit values in these columns by selecting an appropriate cell and typing the new value. You can Cut, Copy and Paste values to and from these columns. If you add numbers to a blank row, a new trial will be generated. The next available trial ID will be automatically added to the first column.

A blank column comes next to mark the end of the set variables.

Any remaining columns correspond to the extract variables (the name of the variable is at the top of the column). These columns are initially empty. When a trial successfully runs, the results of the Extract commands will be added to the appropriate cells.

To save the data in the spreadsheet, either click the **Save** button or select **File→Save** or **File→Save As**. You can save the data in .dat format to view it in TONYPLOT or as space-delimited text that can be imported into a spreadsheet program. Selecting **File→Save As** allows you to define the filename and the file format. Pressing the **Save** button or selecting **File→Save** saves the data to the most recent file defined by **File→Save As**. If you haven't selected **File→Save As**, then pressing the **Save** button or selecting **File→Save** opens the File→Save As Dialog.

The **Keep Deckbuild .out files** check box behaves in the same way as the "log" command for DBINTERNAL. If you check this box, the .out files are not deleted when a trial is finished.

The **Maximum number of concurrent trials** defines the maximum number of child deckbuilds that DBIT can start. If you are running on a computer with more than one processor, or if you have a grid computing engine, you can set this higher than 1.

DBIT is designed to use FLOWTRACER/NC™ by Runtime Design Automation or the N1 GRID ENGINE™ by Sun. If DBIT detects a grid computing engine, you can run the trials on the network rather than the local machine.

B.6.3: The Command Menu

The **Command** menu gives you access to the commands that are used to run the child simulations. In these dialog boxes, the text in angular brackets (such as <in_file> and <silock_command>) are text generated by DBIT, which will be different for each trial. Leave these variables as they are.

"deckbuild..."

The default command to start the child simulation is

```
deckbuild -run <in_file> -int -outfile <out_file>
```

If you want to add any command line options to this command (see Section B.4: "Technical Details"), then add them in this dialog window.

"silock (local host)..." and "silock (grid engine)..."

silock is a small program that monitors the progress of a child trial (started with the deckbuild command). When the trial is finished, silock removes a lock file that allows DBIT to determine the trial has finished. The two versions of the silock command are for when the trials are run directly by DBIT (the local host version) or when the trials are submitted to a grid engine. You should not change these commands. The default command for "silock (local host)..." is

```
silock -f -l <lock_file> -x <deckbuild_command>
```

and the default command for "silock (grid engine)..." is

```
silock -l <lock_file> -x <deckbuild_command>
```

"Flowtracer/NC..." and "Sun N1..."

These are the default commands used to submit a job to the appropriate grid engines.

The default command for FLOWTRACER/NC™ is

```
nc run -E 'SNAPSHOT+D(DISPLAY=[hostname]:0)' <silock_command>
```

The default command for N1 GRID ENGINE™ is

```
qsub -cwd -N <trial_name> -b y "<silock_command>"
```

If you have either of these grid engines, please refer to their documentation for more details on these commands.

This page is intentionally left blank.

C.1: List of Variables

S_EXAMPLES specifies the location of the SILVACO standard examples and overrides the default setting.

<simulator>_ARG overrides the default simulator arguments set in the Simulator Properties popups. The variable below would setup DECKBUILD to start version 1.0.0.A of ATHENA by default.

```
setenv ATHENA_ARG "athena -V 1.0.0.A"
```

<simulator>_HOST overrides the default simulator host setting in the Simulator Properties popups. The variable below would setup DECKBUILD to start ATLAS on hostname `sgi3` by default.

```
setenv ATLAS_HOST sgi3
```

DB_MESH_DIR sets the template directory used in the ATHENA Adaptive Meshing popup. The template files are read and displayed from the directory specified by this variable.

DB_SYSTEM_OPTION enables system commands for use in DECKBUILD and VWF AUTOMATION TOOLS.

NICE_ARG sets the simulator nice value for use in DECKBUILD and VWF AUTOMATION TOOLS.

DB_REMOTE_DIR sets the remote simulation tmp directory, which must be mounted on the host executing DECKBUILD and the host executing the simulator

DB_REMOTE_CMD specifies the remote shell command to be used for remote simulation.

DB_REMOTE_STRIP specifies the automount prefix (usually `/tmp_mnt`) to be removed from paths for remote simulation.

This page is intentionally left blank.

D.1: Text Subwindow Error Messages

The error message "Insertion failed" may occasionally pop up when building very large decks. This means that so many edits were made to the deck that the text subwindow can't handle any more changes. This only happens when building a deck that has not been saved to a file yet. Otherwise, the text editor automatically saves the changes to the file as necessary.

The cure and the prevention are the same: if this error occurs, simply save the deck to a file. An alternate preventative measure is to add a line of the form:

```
text.maxDocumentSize: N
```

to the `.Xdefaults` file, where `N` is the number of bytes allowed before `textedit` saves the file (2000 by default). Try using `N` set to 20000.

D.2: TTY Subwindow Error Messages

The *tty* subwindow, like the text subwindow, has a limit on how big it can grow before an error occurs. In rare circumstances, it is possible to overflow the *tty* subwindow. But it takes a great deal of simulation output to do it (many thousands of lines). If an error message such as “pty: insertion failed” appears, either clear the contents of the *tty* subwindow, or turn scrolling off (because the log is only saved if scrolling is turned on). If this error comes up repeatedly, the best solution is to disable scrolling. Disable scrolling by invoking the **tty** menu in the *tty* subwindow and selecting **Disable Scrolling**.

A

Adding Targets	
Curve Values	6-14-16
Point Values	6-14
<i>See also</i> Targets	
Advanced Features (Automation Tools)	
Worksheets	6-24-25
Advanced Topics	
Extraction	1-3
Generic Decks	1-3
ATHENA	3-21
<i>See also</i> Process Simulators	
Auto Interfacing	3-30-32

B

Bipolar Extract	
QUICKBIP	5-40-42
BJT	5-32
<i>See also</i> Device Extraction	
Breakdown Voltage Calculation	A-6

C

Calculation	
Breakdown Voltage	A-5-6
Sheet Resistance	A-4
Threshold Voltage	A-5
Clever	3-22
Commands	
Clever	3-22
Parsing	3-20
Process Simulators. <i>See</i> Process Simulators	
<i>See also</i> Statements	
Concentration Dependent Mobility	A-2
Controls	
Main Window	3-4-5
Text Subwindow	3-5-7
TTY Subwindow	3-8-9
Curved Target	
Creating	6-14-15
Creating from a Data File	6-15-16
Curves	
Abs Operator with Axis	5-34
Ave Operator	5-33
Axis Manipulation Combined with Max and Abs Operators	5-35
Axis Manipulation Combined with Y Value Intercept	5-35
Axis Manipulation with Constants	5-34
Creation	5-33
Data Format File Extract with X Limits	5-35
Derivative	5-35
Gradient at Axis Intercept	5-34
Impurity Transform against Depth	5-35

Max Operator	5-33
Max Operator with Axis Intercept	5-34
Min Operator	5-33
Min Operator with Axis Intercept	5-34
Second Intercept Occurrence	5-34
X Axis Interception of Line Created by Maxslope Operator	5-34
X Value Intercept for Specified Y	5-33
Y Axis Interception of Line Created by Minslope Operator	5-35
Y Value Intercept for Specified X	5-33
Customized Extract Statements	
Defaults	5-20-21
Syntax	5-7-20
<i>See also</i> Extract	
Cutline	
Loading	3-25-26

D

DBInternal	B-6
Commands. <i>See also</i> DBInternal Commands	
DBIT. <i>See also</i> DBIT	
Example	B-1–B-2
Experiment File	B-4
Technical Details	B-5
Template File	B-3
DBInternal Commands	
doe	B-6–B-7
endsave	B-8
log	B-8
monte_carlo	B-8–B-10
no_exec	B-10–B-11
option	B-11
save	B-11–B-12
sweep	B-12–B-13
DBIT	B-14–B-19
<i>See also</i> DBInternal	
Deck Writing Paradigm	3-20
Default Simulator	3-3
Design of Experiments (DOE)	
box_behnken	B-7
circumscribed_circle	B-7
face_centered_cubic	B-7
gradient_analysis	B-7
three_level_full_factorial	B-7
two_level_full_factorial	B-7
two_level_half_factorial	B-7
<i>See also</i> doe Command	
Device Extraction	
BJT	5-32
Curve	5-29-30
Curve Manipulation	5-31-32
Device Extraction	5-29-32
<i>See also</i> Extract	

E

Electrodes 3-34-35
Environment Variables C-1
Error Messages. *See* Text and TTY Subwindows

Execution Control

Breakpoints 3-18
Buttons 3-18
Concepts 3-17
Initializing the Simulator 3-19
Pausing, Stopping, and Restarting the Simulator 3-19
Setting Lines 3-18
Stepping Through and Running the Deck 3-18

Extract

Customized Statements 5-7-28
Device Extraction 5-29-32
Features 5-38-39
MOS Device Tests 5-36
Process Extraction 5-2-6
QUICKBIP Bipolar Extract 5-40-42
Results 5-37
Using with ATLAS 5-43-44

Extract Features

Extract Name 5-38
Extraction and the Database (VWF) 5-39
Min and Max Cutoff Values 5-38
Multi-Line Extract Statements 5-39
Variable Substitution 5-38

F

Features

Advanced Topics 1-3
Auto-Interface 1-2
See also Auto-Interfacing
Examples and Tutorial 1-3
See also Quickstart
Execution Control 1-3
Optimization 1-3-4
Simulators 1-2

Field Dependent Mobility Model A-3

File I/O

Creating 6-26-27
Saving 6-27
Working With Existing File 6-27

Flowtracer/NC™ B-18, B-19

G

General Curve Examples 5-33-35
See also Curves

H

History Function

Controlling 3-28

I

IC Layout Interface

Creating Decks 3-33-34
Mask Bias, Misalignment, and Delta CD 3-36
Regions 3-34-36
Rules of Thumb 3-36
Using DevEdit 3-37

Initializing 3-19

Internal Interface 3-47

M

Main DeckBuild Controls

Arguments 3-16
Choosing a Simulator 3-10-11
Control Pad 3-10
Execution Control 3-17-19
Formatting 3-16
Messages 3-15-16
Options 3-13-15
Simulator Controls 3-11-12
Simulator Properties 3-11
Start Simulator 3-11

Main Window

Execution Control Buttons 3-5
Menu Buttons 3-4

Mask

Misalignment and CD Experimentation 4-14
Statements 3-33

MaskViews 4-15

Starting 3-24-26

MOS Device Tests 5-36

N

N1 Grid Engine™ B-18, B-19

O

Optimizer 1-3-4, 6-1-30

Features 6-1
File I/O 6-26-27
Graphics 6-21-22
Optimization Modes 6-3
Optimization Tuning 6-30
Optimizer Window 6-2
Parameters 6-4-11
Printing 6-28-29
Results 6-23
Setup 6-20
Targets 6-12-19
Terminology 6-1
Worksheet Editing 6-24-25

Optimizer Parameter Editing

Numeric Values 6-7
Parameter Name 6-8
Response Type 6-7

Optimizer Parameters

Adding 6-4-6
Copying 6-10
Defaults 6-10
Deleting 6-6-7

Editing	6-7-8
Enabling/Disabling	6-10-11
Folding Columns	6-11
Linked	6-9-10
Optimizer Results	
Sensitivity	6-23
Optimizer Setup	
Editing	6-20
Saving	6-20

P

Physical Models	A-1
Process Extraction	
Curves	5-5-6
Entering Statements	5-4-5
Examples	5-21-28
Process Extraction Examples	
1D Material Region Boundary	5-24
1D Max/Min Concentration	5-23
2D Concentration Area	5-24
2D Concentration File	5-24
2D Material Region Boundary	5-24
2D Max/Min Concentration	5-23
2D Maximum Concentration File	5-24-25
ED Tree (Optolith)	5-28
Elapsed time	5-28
Electrical Concentration Curve	5-28
Junction Breakdown Curve	5-26
Junction Capacitance Curve	5-25-26
Junction Depth	5-21
Material Thickness	5-21
QUICKMOS 1D Vt	5-22
QUICKMOS CV Curve	5-25
Sheet Conductance	5-22
Sheet Resistance	5-22
Sheet Resistance/Conductance Bias Curves	5-27-28
SIMS Curve	5-26
SRP Curve	5-27
Surface Concentration	5-21
Process Simulators	
Selecting Categories	3-22
Writing Process Input Decks	3-21-22
Writing Text	3-22
Purpose	1-1

Q

QUICKBIP	
Bipolar Extract	5-40-42
<i>See also</i> Extract	
QuickStart	2-1-6
Quitting	2-6

R

Random Distributions	
gamma	B-10
log_normal	B-9
normal	B-9

uniform	B-9
Weibull	B-10
<i>See also</i> monte_carlo Command	
Regions	
Electrodes	3-34-35
Enabling	3-36
Remote Simulation	
Options	3-48
Troubleshooting	3-48-49
Running Decks	
Plotting The Current Structure	2-5
Resetting The Current Line	2-4
Saving	2-5

S

Sheet Resistance Calculation	A-4
SmartSpice	
Interface	3-46
SSUPREM3	2-1-3
Starting	2-1, 3-1-3
Statements	
ASSIGN	4-2-4-4
AUTOELECTRODE	4-5
DEFINE	4-6-4-7
ELSE	4-11
EXTRACT	4-8
GO	4-9-4-10
IF	4-11
IF.END	4-11
L.END	4-12-4-13
L.MODIFY	4-12-4-13
LOOP	4-12-4-13
MASK	4-14-4-15
MASKVIEWS	4-16
SOURCE	4-20-4-21
STMT	4-22
SYSTEM	4-23
TONYPLOT	4-24
UNDEFINE	4-6-4-7
Subwindow. <i>See</i> Text and TTY Subwindows	

T

Target Editing	
Numeric Values	6-17
Target Name	6-18
Target Type	6-17
Targets	
Adding	6-12-16
Deleting	6-16-17
Editing	6-17-18
Enabling/Disabling	6-18
Folding Columns	6-19
<i>See also</i> Optimizer	
Text Editor	
Starting	3-27
Text Subwindow	
Adding And Deleting Text	3-7

Copying Text	3-7
Creating Files	3-5-7
Cutting And Pasting	3-7
Editing Input Decks	3-7
Error Messages	D-1
Loading Files	3-6
Saving Changes	3-7
See also Controls	
Threshold Voltage Calculation	A-5
TonyPlot	
Starting	3-23-24
Tools	
Manager	3-27
Maskviews	3-24-26
See also MaskViews	
Text Editor	3-27
TonyPlot	3-23-24
See also TonyPlot	
TTY Subwindow	
Cut, Paste, and Copy	3-8
Editing	3-8
Entering Commands	3-8
Error Messages	3-9, D-2
Saving/Resetting	3-9
Tutorial . See Quickstart	

U

UTMOST

Input Deck	3-38-45
Interface	3-38

W

Worksheet

Mouseless Operation	6-25
Numeric Values	6-24
Selecting Rows	6-24
See also Optimizer	
Writing SSUPREM3 Input Deck	2-1-3